APPENDIX A: DETAILED SYSTEM DESCRIPTIONS AND EVALUATIONS

A.1 Introduction

For each of the systems considered in this study (LSF, PBS, Condor, RES, Compaq DCE, Sun Grid Engin/CODINE, Globus, Legion, NetSolve, MOSIX, AppLES, and NWS), the appendix provides a detailed description and evaluation. The evaluation is based on the requirements of section 2 and follows a scoring scheme that attaches a weight of 4 points to each of the major requirements. The total number of major requirements is 25 and the total points, is therefore, 100. A score of 0 was assigned uniformly in any case where it is not known whether the corresponding requirement is met by the examined system. We also acknowledge that information available on these systems could be thought of as limited given the level of depth that is of interest to this study. Therefore, we anticipate some limited inaccuracies, however, due to the large number of requirements considered and the uniform treatment that we attempted, we believe that the overall relative scores of these systems, and thus the ranking, should be considered accurate.

A.2 LSF (Load Sharing Facility)

Background information

A. Authors

Platform Computing Corporation

B. Support

Platform Computing Corporation

C. Distribution (commercial/public domain)

Commercial domain

D. License

Commercial license

System description

E. Objective/ type of the system

a. centralized Job Management System

LSF is a loosely coupled cluster solution for heterogeneous systems. LSF is built in layers, the base system services provide dynamic load balancing & transparent access to the resources available on all machines participating in the cluster. The base services are much like operating system services but they are cluster-based. The LSF Batch system is built on the cluster system services and provides a central, scalable, fault tolerant batch system.

F. Major modules

a. User server

LSF Batch commands&API + "mbatchd" daemon

b. Job scheduler

LSFJobScheduler + "mbatchd"

c. Resource manager

This module consists typically of the two functional units:

- resource monitor

LIM + MLIM

- job dispatcher

"mbatchd" + "sbatchd"

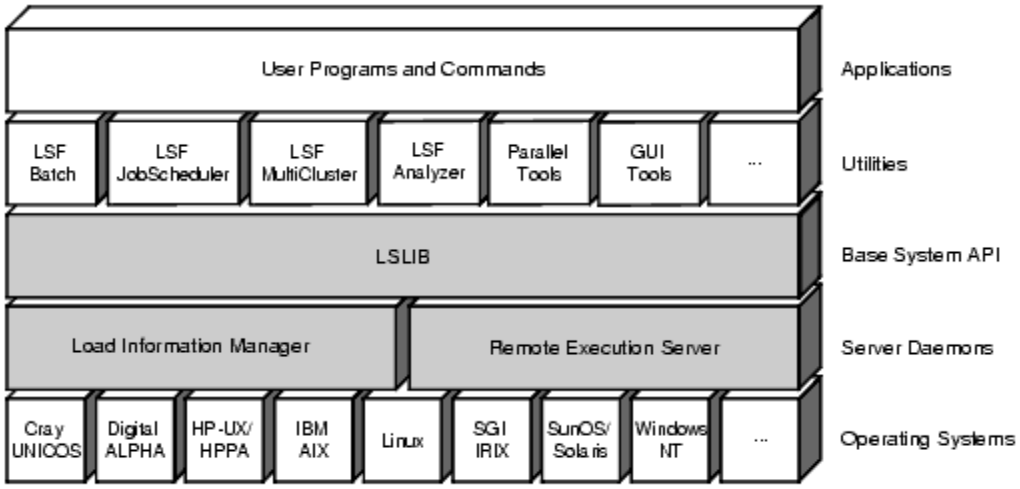
d. checkpointing module

LSF provides support for most checkpoint and restart implementations through uniform interfaces, echkpnt and erestart.

LSF layered architecture

LSF Standard Edition consists of *LSF Base* and *LSF Batch*.

Structure of LSF Base



The software modules that make up LSF Base are shaded.

server = host that runs load shared jobs.

Load Information Manager (LIM) + Remote Execution Server (RES) = daemons running on every server.

LIM provides convenient services that help job placement, host selection, and load information that are essential to the scheduling of jobs. `lsrun` and `lsgrun`, for example, use the LIM's placement advice to run jobs on the least loaded yet most powerful hosts. When LIM gives placement advice, it takes into consideration many factors, such as current load information, job's resource requirements, and configured policies in the LIM cluster configuration file.

RES provides transparent and efficient remote execution and remote file operation services so that jobs can be easily shipped to anywhere in the network once a placement decision has been made. Files can be accessed easily from anywhere in the network using remote file operation services.

LSLIB - LSF Base system API allows users to write their own load-sharing applications on top of LSF Base.

LSF Base provides basic load-sharing services across a heterogeneous network of computers. It is the base software upon which all other LSF products are built. It provides services such as resource information, host selection, placement advice, transparent remote execution, and remote file operation.

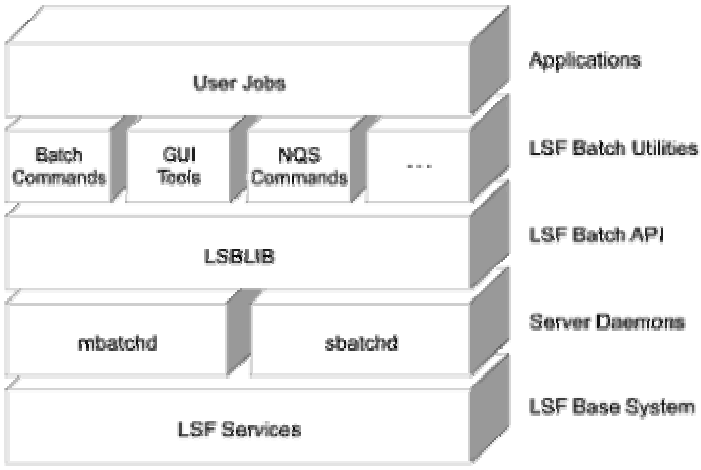
If sophisticated job scheduling and resource allocation policies are necessary, more complex scheduling must be built on top of LSF Base, such as LSF Batch. Since the placement service from LIM is just advice, LSF Batch makes its own placement decisions based on advice from LIM as well as further policies that the

site configures. LSFJobScheduler adds calendar and event processing services to the LSF Batch architecture and job processing mechanism. Use LSBLIB to access LSF JobScheduler functionality.

Structure of LSF Batch

LSF Batch is a distributed batch system built on top of LSF Base to provide powerful batch job scheduling services to users. The services provided by LSF Batch are extensions to the LSF Base services.

The following diagram shows the components of LSF Batch and the interactions among them.



LSF Batch accepts user jobs and holds them in queues until suitable hosts are available. Host selection is based on up-to-date load information from the *master LIM*, so LSF can take full advantage of all your hosts without overloading any.

There is one master batch daemon (MBD) running in each LSF cluster, and one slave batch daemon (SBD) running on each LSF server host. User jobs are held in queues by MBD, which checks the load information on all candidate hosts periodically. When a host with the necessary resources becomes available, MBD sends the job to SBD on that host for execution. When more than one host is available, the best host is chosen. SBD controls the execution of the jobs and reports job status to MBD.

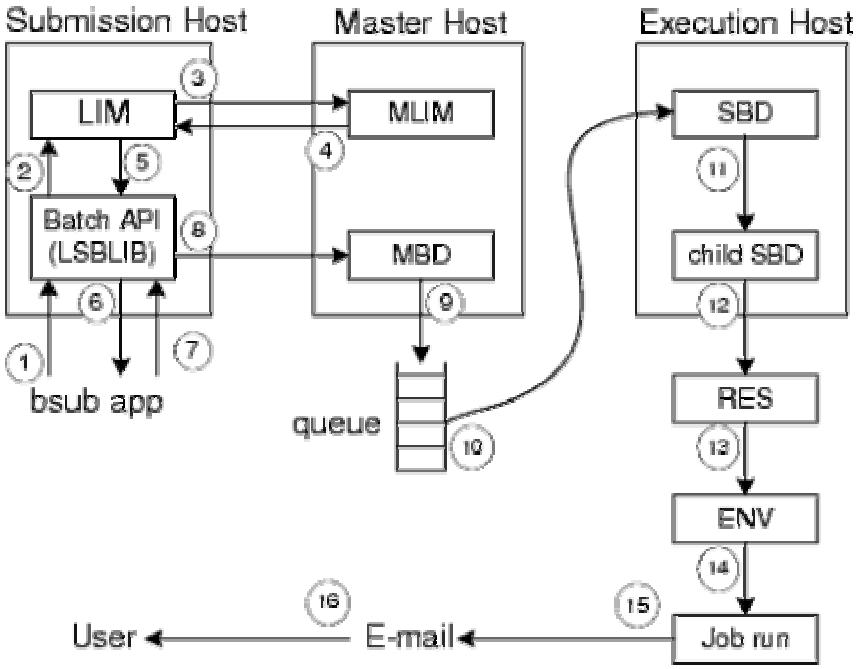
LSF Batch has the same view of cluster and master host as the LSF Base, although LSF Batch may only use some of the hosts in the cluster as servers. SBD runs on every host that the LSF administrator configures as an LSF server. **MBD always runs on the same host as the master LIM**.

The LSF Batch Library (LSBLIB) is the Application Programming Interface (API) for LSF Batch, providing easy access to the services of MBD and SBD. LSBLIB provides a powerful interface for advanced users to develop new batch processing applications in C.

G. Functional Flow

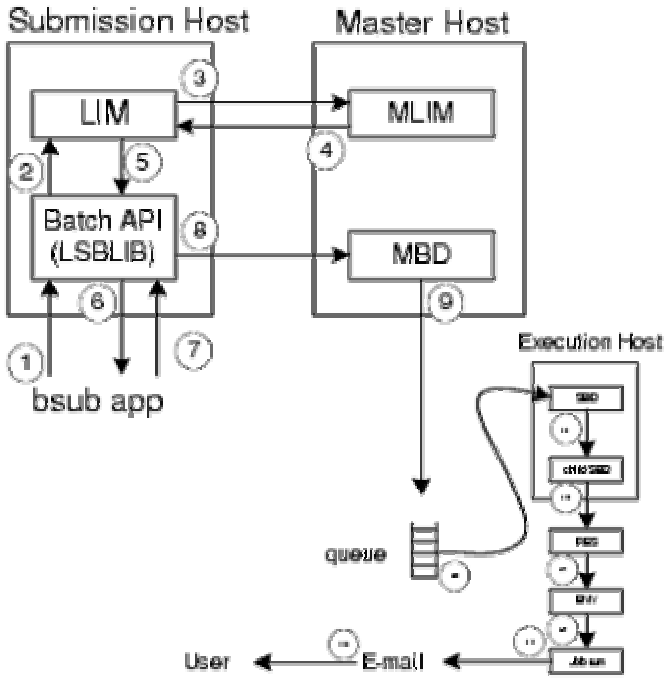
Application and LSF Batch Interactions

LSF Batch operation relies on the services provided by LSF Base. LSF Batch contacts the master LIM to get load and resource information about every batch server host. The diagram below shows the typical operation of LSF Batch:



LSF Batch executes jobs by sending user requests from the submission host to the master host. The master host puts the job in a queue and dispatches the job to an execution host. The job is run and the results are emailed to the user.

Unlike LSF Base, the submission host does not directly interact with the execution host.



1. `bsub` or `lsb_submit()` submits a job to LSF for execution.

2. To access LSF base services, the submitted job proceeds through the LSF Batch library (LSBLIB) that contains LSF Base library information.

3. The LIM communicates the job's information to the cluster's master LIM. Periodically, the LIM on individual machines gathers its 12 built-in load indices and forwards this information to the master LIM.

4. The master LIM determines the best host to run the job and sends this information back to the submission host's LIM.

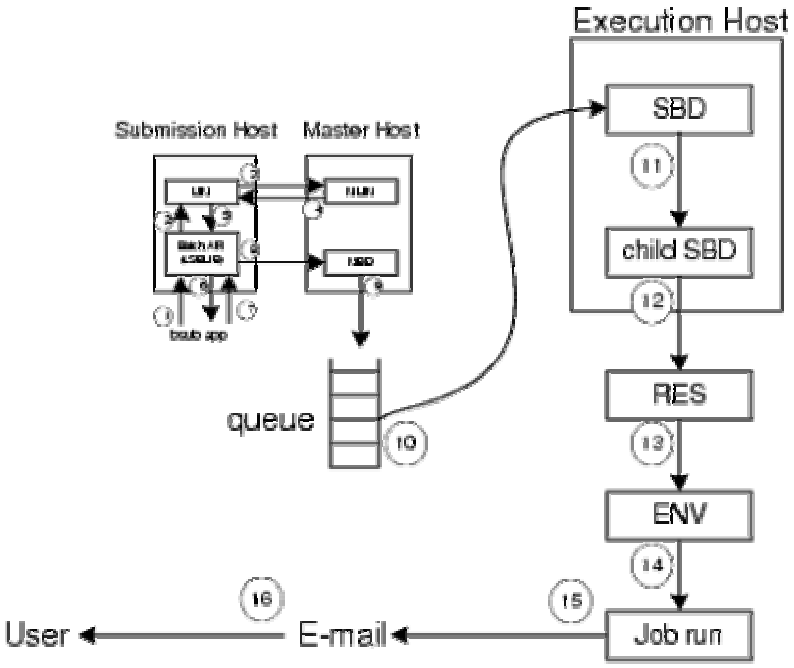
5. Information about the chosen execution host is passed through the LSF Batch library.

6. Information about the host to execute the job is passed back to `bsub` or `lsb_submit()`.

7. To enter the batch system, `bsub` or `lsb_submit()` sends the job to LSBLIB.

8. Using LSBLIB services, the job is sent to the `mbatchd` running on the cluster's master host.

9. The `mbatchd` puts the job in an appropriate queue and waits for the appropriate time to dispatch the job. User jobs are held in batch queues by `mbatchd`, which checks the load information on all candidate hosts periodically.



10. The `mbatchd` dispatches the job when an execution host with the necessary resources becomes available where it is received by the host's sbatchd. When more than one host is available, the best host is chosen.

11. Once a job is sent to an sbatchd, that sbatchd controls the execution of the job and reports the job's status to `mbatchd`. The sbatchd creates a child sbatchd to handle job execution.

12. The child sbatchd sends the job to the RES.

13. The RES creates the execution environment to run the job.

14. The job is run in the execution environment.

15. The results of the job are sent to the email system.

16. The email system sends the job's results to the user.

`The mbatchd` always runs on the host where the master LIM runs. The sbatchd on the master host automatically starts the `mbatchd`. If the master LIM moves to a different host, the current `mbatchd` will automatically resign and a new `mbatchd` will be automatically started on the new master host.

The log files store important system and job information so that a newly started `mbatchd` can restore the status of the previous `mbatchd`. The log files also provide historic information about jobs, queues, hosts, and LSF Batch servers.

H. Distribution of control

Central control

I. Distribution of information services

Master LIM (load information manager) is a central manager, which maintains load information from all servers and dispatches all cluster jobs, for large clusters, this single master policy is inadequate. LSF divides the large cluster into a number of smaller subclusters. There is still a single master LIM within a subcluster, but the master LIMs exchange load information and collectively make inter-subcluster load-sharing decisions.

J. Communication architecture

LSF uses UDP and TCP ports for communication. All hosts in the cluster must use the same port numbers so that each host can connect to servers on other hosts.

b. Shared file systems

It is preferred to have a uniform path, account (although there is also an account mapping mechanism), and shared file system, although LSF can do without these. LSF manages user permissions for NFS, **AFS**, and DFS accesses, so users can use LSF no matter what type of file system their files are stored on. The choice of installation directory for LSF does not affect user access to load sharing.

LSF MultiCluster extends the capabilities of LSF Standard Edition by sharing the resources of an organization across multiple cooperating LSF clusters. Load-sharing and batch processing happens not only within the clusters, but also among them. Resource ownership and autonomy is enforced, **non-shared user accounts and file systems are supported**, and communication limitations among the clusters are also considered in job scheduling. LSF MultiCluster allows cluster sizes to grow to many thousand hosts and enables geographically separate locations to distribute jobs across clusters.

K. Interoperability

a. JMS running as a part of another system

Globus

Globus can submit jobs to LSF.

b. JMS using another system in place of its module

Condor

LSF is integrated with the checkpointing capabilities of Condor. This integration allows jobs submitted to LSF to use the native Condor checkpointing facilities to perform user-level job checkpointing. Condor checkpointing is supported on many platforms including Linux. The Condor checkpoint support files are installed in place of the LSF checkpointing files. The support files are available as a separate distribution from Platform Computing. After Condor checkpoint support is installed, jobs are built, re-linked, checkpointed, and restarted the same way as user-level checkpointable job are in LSF.

SNMP

To integrate with existing network and system management frameworks, LSF supports SNMP, an IETF (Internet Engineering Task Force) standard protocol used to monitor and manage devices and software on the network. Platform Computing has also defined a Management Information Base (MIB) specific to LSF. Any SNMP client, from command-line utilities to full network and system-management frameworks, can monitor information provided by the LSF SNMP agent.

NQS

NQS interoperation with LSF allows LSF users to submit jobs to remote NQS servers using the LSF user interface. The LSF administrator can configure LSF queues to forward jobs to NQS queues. Users may then use any supported interface, including LSF commands,

lsNQS commands, and xlsbatch, to submit, monitor, signal and delete batch jobs in NQS queues. This feature provides users with a consistent user interface for jobs running under LSF and NQS.

Major requirements

1. **Availability**
   - **a. binary code** — **Yes***(demo + eval + commercial versions)*
   - **b. source code** — **No**
   - **c. documentation** — **Yes**
   - **d. roadmap** — **Yes**
   - **e. training** — **Yes**
   - **f. customer support** — **Yes**

   Score: *3 (no source code)*

2. **Operating systems/platforms**
   - **a. submission** — **UNIX/NT**
   - **b. execution** — **UNIX/NT**
   - **c. control** — **UNIX/NT**

   Score: *4*

3. **Batch job support**

   Score: *4*

4. **Interactive job support**

   Score: *4*

5. **Parallel job support**

Standard parallel job support:

LSF provides a generic interface to parallel programming packages so that any parallel package can be supported by writing shell scripts or wrapper programs. On SGI IRIX 6.5 systems, LSF can be configured to make use of IRIX cpusets to enforce processor limits for LSF jobs.

Parallel Job Processor Reservation - Running parallel applications and sequential applications in the same LSF cluster potentially starves out the parallel applications in favor of the sequential applications. A sequential batch job typically uses one processor while a parallel job will use more than one processor. Parallel job processor reservation allows a queued parallel job to reserve processors (job slots) for a specified time period.

LSFParallel:

LSF Parallel supports programming, testing, execution and runtime management of parallel applications in production environments. LSF Parallel is integrated with the LSF Batch system to provide fine-grained resource control over parallel applications using vendor supplied MPI libraries. Parallel jobs can be submitted into the LSF Batch system and LSF Parallel will automatically keep track of the parallel application, provided the application is compiled with the vendor MPI libraries.

LSF Parallel provides the following features:
Dynamic resource discovery and allocation
Transparent invocation of the parallel components
Full job control of the distributed parallel components
The standard MPI interface
Operates on major UNIX platforms

Score: *4*

6. Resource requests by users

Resource requirement strings
Load indices (for a server; updated by LIM):

- run queue length
- CPU utilization
- paging activity
- logins
- idle time
- available swap space
- available memory
- available space in temporary file system
- disk I/O
- external load index configured by LSF administrator

Static resources:

- host type
- host model
- host name
- CPU factor (relative)
- host can run jobs (boolean)
- number of processors
- number of local disks
- maximum RAM memory available to users
- maximum available swap space megabytes LIM
- maximum available space in temporary file system

Shared resources

- floating licenses for software packages
- disk space on a file server which is mounted by several machines
- the physical network connecting the hosts

Possibility of defining your own resource type.

Score: *4*

7. Limits on resources by administrators

a. general *Yes*

Resource limits are constraints you or your LSF administrator can specify to limit the use of resources. Jobs that consume more than the specified amount of a resource are signaled or have their priority lowered. Resource limits can be specified either at the queue level by your LSF administrator (lsb.queues) or at the job level when you submit a job. Resource limits specified at the queue level are hard limits while those specified with job submission are soft limits.
Queue-Level Resource Limits (1 limit / 2 limits = hard + soft)

- RUNLIMIT
- CPULIMIT

- DATALIMIT
- MEMLIMIT
- PROCESSLIMIT
- PROCLIMIT

Job-Level Resource Limits (with bsub *command)*

- CPU Time Limit
- Run Time Limit
- File Size Limit
- Data Segment Size Limit
- Stack Segment Size Limit
- Core File Size Limit
- Memory Limit

IRIX 6.5.8 supports job-based limits. LSF can be configured so that jobs submitted to a host with the IRIX job limits option installed are subject to the job limits configured in the IRIX User Limits Database (ULDB).

b. specific users or groups *Yes*

The user must be authorized to execute commands remotely on the host.

Score: *4*

8. File stage-in and stage-out

RES module.

Score: *4*

9. Flexible scheduling

a. several scheduling policies *Yes*

LSF provides several scheduling policies to manage the batch system. Fairshare policy can be applied at the queue or host partition level to manage conflicting demands to computing resources.
Policies:

- fairshare
- preemptive
- preemptable
- exclusive
- FCFS

An interactive batch job is scheduled using the same policy as all other jobs in a queue. This means an interactive job can wait for a long time before it gets dispatched. If fast response time is required, interactive jobs should be submitted to high-priority queues with loose scheduling constraints.

b. policies changing in time *No info.*

c. scheduling multiple resources simultaneously *Yes*

d. highly configurable *Yes*

LSFJobScheduler adds calendar and event processing services to the LSF Batch architecture and job processing mechanism.

Score: *3*

10. Job priorities

The job owner can change the priority of their own jobs. User-assigned job priority provides controls that allow users to order their jobs in a queue. Automatic job priority escalation automatically increases job priority of jobs that have been pending for a specified period of time.
Jobs are still subject to all scheduling policies regardless of job priority. Jobs with the same priority are ordered first come first served.
MAX_USER_PRIORITY – defined by admin

Score: *4*

11. Timesharing

- **a. processes** *Yes*
- **b. jobs** *Yes*

Score: *4*

12. Impact on owners of computational nodes

A user's job can be rejected at submission time if the submission parameters cannot be validated. Sites can implement their own policy to determine valid values or combinations of submission parameters. The validation checking is performed by an external submission program (esub) located in LSF_SERVERDIR. (validation file = LSB_SUB_PARM_FILE)

Score: *4*

13. Checkpointing

- **a. user-level** *Yes*
- **b. run-time library level** *Yes*
- **c. OS (kernel) level** *Yes (Only for: Cray UNICOS, IRIX 6.4 and later, NEC SX-4 and SX-5)*

Score: *4*

14. Suspending/resuming/killing jobs

- **a. user** *Yes*
- **b. administrator** *Yes*
- **c. automatic** *Yes*

Score: *4*

15. Process migration

- **a. user** *Yes*
- **b. administrator** *Yes*
- **c. automatic** *Yes*

Automatic job migration. Automatic job migration works on the premise that if a job is suspended (SSUSP) for an extended period of time, due to load conditions or any other reason, the execution host is heavily loaded. To allow the job to make progress and to reduce the load on the host, a migration threshold is configured. LSF allows migration thresholds to be configured for queues and hosts. The threshold is specified in minutes.

When configured on a queue, the threshold will apply to all jobs submitted to the queue. When defined at the host level, the threshold will apply to all jobs running on the host. When a migration threshold is configured on both a queue and host, the lower threshold value is used. If the migration threshold is configured to 0 (zero), the job will be migrated immediately upon suspension (SSUSP).

LSF provides different alternatives for configuring suspending conditions. Suspending conditions are configured at the host level as load thresholds, whereas suspending conditions are configured at the queue level as either load thresholds, or by using the STOP_COND parameter in the lsb.queues file, or both. Normally, the swp and tmp indices are not considered for suspending jobs, because suspending a job does not free up the space being used. However, if swp and tmp are specified by the STOP_COND parameter in your queue, these indices are considered for suspending jobs. The load indices most commonly used for suspending conditions are the CPU run queue lengths, paging rate, and idle time. To give priority to interactive users, set the suspending threshold on it load index to a non-zero value. Jobs are stopped (within about 1.5 minutes) when any user is active, and resumed when the host has been idle for the time given in the it scheduling condition.

Score: *4*

16. Static load balancing

Score: *4*

17. Dynamic load balancing

Score: *4*

18. Faul tolerance

a. computational nodes	*Yes*
b. scheduler	*Yes. Disributed scheduler. Copies of transaction files.*

Score: *4*

19. Job monitoring

Logs, monitoring tools, GUIs.

Score: *4*

20. User interface

CLI & GUI

Score: *4*

21. Published APIs

Two libraries available: LSLIB (base library) & LSBLIB (batch library)

Score: *4*

22. Dynamic system reconfiguration

Score: *4*

23. Security

a. Authentication	*Yes (Kerberos, but is not the only one)*
b. Authorization	*Yes*
c. Encryption	*No info.*

Score: *3*

24. Accounting

Each time a batch job completes or exits, LSF appends an entry to the lsb.acct file. This file can be used to create accounting summaries of LSF Batch system use. The bacct command produces one form of summary. The lsb.acct file is a text file suitable for processing with awk, PERL, or similar tools. Additionally, the LSF Batch API supports calls to process the lsb.acct records.

Score: *4*

25. Scalability

Score: *4*

TOTAL SCORE = ***97***

Major disadvantages:

Limited access to source code

Major advantages:

Strong support for parallel jobs
Job migration and dynamic load balancing
Strong checkpointing
Minimal impact on resource owners
GUI to all modules
Good fault tolerance
Strong authenticatian and authorization
High quality of code and documentation

Reference:

http://www.platform.com/index.html
LSF Administrator 's Guide Version 4.1 December 2000
LSF JobScheduler Administrator's Guide
LSF JobScheduler User's Guide
LSF Parallel User 's Guide Version 4.1 December 2000
LSF Programmer 's Guide Version 4.1 December 2000
LSF Reference Guide Version 4.1 December 2000

A.3 Sun Grid Engine/Codine

Background information

A. **Authors**
Gridware Inc.
B. **Support**
Sun Microsystems
C. Distribution (commercial/public domain)
Free/Commercial
D. License
Sun Binary Code License, announced future switch to Open Source license

System description

E. Type of the system
Centralized job management system
F. Major modules
Architecture:

- Master host (*job scheduler* with *resource manager*)
  A single machine selected to perform the following functions:
  - Mater functions: service-level capabilities for dispatching jobs, gathering machine load reports, answering user queries, and accepting and controlling user jobs
  - Scheduling functions: executive-level capabilities for continuous analysis of pending jobs and available resources, job placement, and error-condition handling.

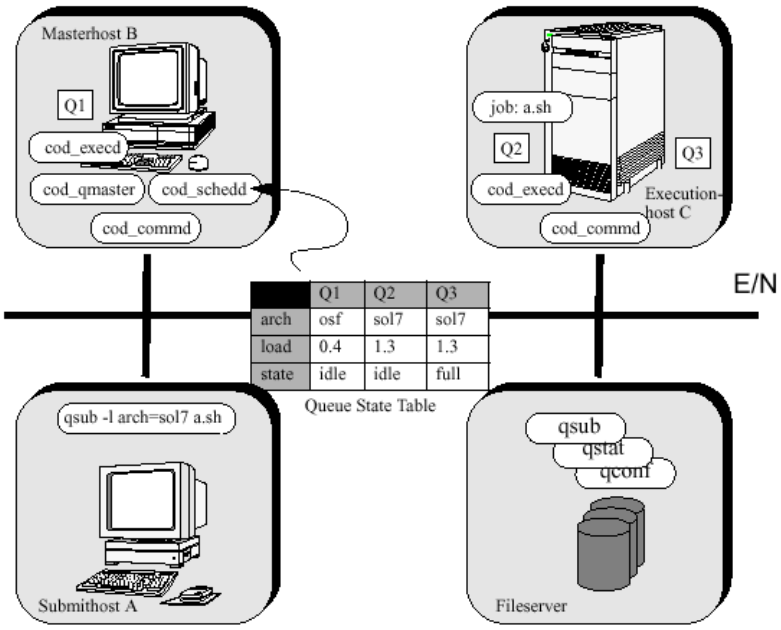
Other machines in the cluster can be designated as "shadow master" machines. A shadow master daemon will monitor the master machine, and if the master goes down, the shadow master will take over as master. This eliminates the master machine as a single point of failure, making the cluster more fault tolerant.

- Execution Host (*job dispatcher*)
  The machines in the cluster that are eligible to execute jobs are called execution hosts. Execution hosts run the Commd and the Execd daemons. As summarized in the table above, the Execd monitors load on the execution host. This load information includes CPU load, swap, and memory information. In addition, any load quantity that can be measured can be easily added to the load information gathering mechanism. Thus, site specific load information, such as the availability of a certain license, network bandwidth, or local disk space can be added. These resources can then be requested by Sun Grid Engine software users. Execution hosts can be configured to be submitting hosts as well.
- Submit Host (*user server*)
  Machines may be configured to not run jobs, but to only submit jobs. This type of host is called a submit host. There are no daemons needed for a submit host. The only requirement for a submit host is that it be added to the list of

eligible submit hosts in the cluster. This is designed to be able to control access to Sun Grid Engine software.

- Administration Host
  Certain administrative functions, such as changing queue parameters, adding new nodes, or adding or changing users must be done by an administrator on an administration host. As with the submit host, no special daemons are needed to be an administration host. The administration host must be added to the list of eligible administration hosts in the cluster. This is also used for Sun Grid Engine software security. For example, administration hosts can be selected to be only those hosts in a secure computer room.

G. Functional flow



Job flow:

- Job Submission
  When a user submits a job from a submit host, the job submission request is sent to the master host. The submit client retrieves the user identity (interfaces to authentication utilities), and the current working directory (cwd) is transferred if the cwd-flag is activated. The master host acknowledges receipt of the request, saves the job to the database, and notifies the scheduler.
- Job Scheduling
  The scheduler determines which queue the job will be assigned to. It assesses the load, checks for licenses, and any other job requirements. It continues to try to schedule until the job is dispatched. The scheduler then returns its choice of queue to the Sun Grid Engine software master.
- Job Execution
  Upon obtaining scheduling information from the scheduler, the master then sends the job to an execution host to be executed. Before the job is started, the execution daemon and the job shepherd perform the following tasks:
  - Change to the appropriate working directory

- Set the environment
- Set the array session handle (on supported operating systems)
- Set the processor set (on supported operating systems)
- Change the UID to the job owner
- Set the resource limits
- Retrieve the accounting information

The execution daemon saves the job to the job information database, and starts the shepherd process, which starts the job and waits for completion. When the end of job is reported to the master, the execution daemon removes the job from the database. The master host then updates the job accounting database.

H. Distribution of control

Central scheduler with backups

I. Distribution of information services

Information is collected and stored by the scheduler

J. Communication architecture

a. Protocols

TCP/IP

b. Shared file systems

May use NFS

K. Interoperability

a. JMS running as a part of another system

May accept jobs submitted by other systems

b. JMS using another system in place of its module

May take advantage of check pointing libraries from other systems (e.g. Condor's check pointing library), works with Global Resource Director

c. JMS running concurrently with another system

No

Major requirements

1. Availability

a. Binary code — Yes
b. Source code — No/Yes (in future)
c. Documentation — Yes
d. Roadmap — Releasing source code, port entire JMS to other platforms (including Windows NT), release Global Resource director,
e. Training — Yes
f. Customer support — Yes

Score: *3*

2. Operating systems/platforms

a. **Submission** Solaris 2.6, 7, 8, Linux, True64UNIX 5.0, HP-UX 11.0, IBM AIX 5.0, SGI Irix 6.5, planned NT port

b. **Execution** Solaris 2.6, 7, 8, Linux, True64UNIX 5.0, HP-UX 11.0, IBM AIX 5.0, SGI Irix 6.5, planned NT port
c. **Control** Solaris 2.6, 7, 8, Linux, True64UNIX 5.0, HP-UX 11.0, IBM AIX 5.0, SGI Irix 6.5, planned NT port

Score: *3*

3. Batch job support

Yes

Score: *4*

4. Interactive job support

Yes

Score: *4*

5. Parallel job support

Yes

Score: *4*

6. Resource requests by users

Yes

Score: *4*

7. Limits on resources by administrators

a. **General** Yes
b. **Specific users or groups** Yes

Score: *4*

8. **File stage-in and stage-out**

Score: *0*

9. Flexible scheduling

a. Several scheduling policies Yes
b. Policies changing in time Yes
c. Scheduling multiple resources simultaneously Yes
d. Highly configurable scheduling Yes

Score: *4*

10. Job priorities

Policies changing in the range from –1024 to 1023, both inter-queue and intra-queue.

Score: *4*

11. Timesharing

a. Processes Yes
b. Jobs Yes

Score: *4*

12. Impact on owners of computational nodes

Users have full control over their workstations

Score: *4*

13. **Check pointing**

a. User-level Using foreign check pointing libraries or application's own check pointing mechanisms
b. Run-time library level Yes
c. OS (kernel) level Yes (using mechanisms provided by OS on execution host)

Score: *1.5*

14. Suspending/resuming/killing jobs

a. User Yes
b. Administrator Yes
c. Automatic Yes

Score: *4*

15. **Process migration**

a. User Yes
b. Administrator Yes
c. Automatic Yes

Score: *4*

16. Static load balancing

Yes

Score *4*

17. Dynamic load balancing

Score: *4*

18. Fault tolerance

a. Computational nodes Yes
b. Scheduler Yes

Score: *4*

19. Job monitoring

Score: *4*

20. User interface

CLI&GUI to all modules

Score: *4*

21. Published APIs

Score: *4*

22. Dynamic system reconfiguration
Score: *4*

23. Security

a.	Authentication	Basing on user names and ID's
b.	Authorization	Yes (ACL's)
c.	Encryption	No

Score: *2*

24. Accounting
Score: *4*

25. Scalability
Score: *4*

MAXIMUM TOTAL SCORE = **89.5**

Advantages

- Parallel job support
- Job migration support and dynamic load balancing
- Flexible scheduling
- Minimal impact on owners of computational nodes
- GUI to all modules
- Good authorization mechanism

Disadvantages

- No source code available at the moment
- No Windows NT support
- No stage-in nor stage-out mechanisms
- Externally supported check pointing only

References:

Homepage: `http://www.sun.com/software`

The Sun Grid Engine manual
`http://www.sun.com/software/gridware/docs/gridengine-manual.pdf`

Sun Grid Engine Frequent Asked General Questions list
`http://www.sun.com/software/gridware/faqs/faqs.html`

Sun Grid Engine Frequent Asked technical Questions list
`http://supportforum.Sun.COM/gridengine/`

Sun Grid Engine Overview http://www.sun.com/software/gridware/ds-gridware/

Sun Grid Engine Detailed View
http://www.sun.com/software/gridware/details.html

A.4 PBS (Portable Batch System)

Background information

A. Authors

Veridian Systems

B. Support

Veridian Systems

C. Distribution (commercial/public domain)

Comercial domain

D. License

OpenPBS is available for free with a software license agreement (not for commercial use or distribution).

PBSPro can be purchased for $2500 + $100 per CPU (includes source code) => commercial license.

System description

E. Objective/ type of the system

a. centralized Job Management System

It is a batch job and computer system resource management package. It was developed with the intent to be conformant with the POSIX 1003.2d Batch Environment Standard. As such, it will accept batch jobs, a shell script and control attributes, preserve and protect the job until it is run, run the job, and deliver output back to the submitter. PBS may be installed and configured to support jobs run on a single system, or many systems grouped together.

F. Major modules

Job Server (a. user server)

The Job Server is the central focus for PBS. It is generally referred to as *the Server* or by the execution name *pbs_server* .All commands and the other daemons communicate with the Server via an IP network. The Server's main function is to provide the basic batch services such as receiving/creating a batch job, modifying the job, protecting the job against system crashes, and running the job (placing it into execution).

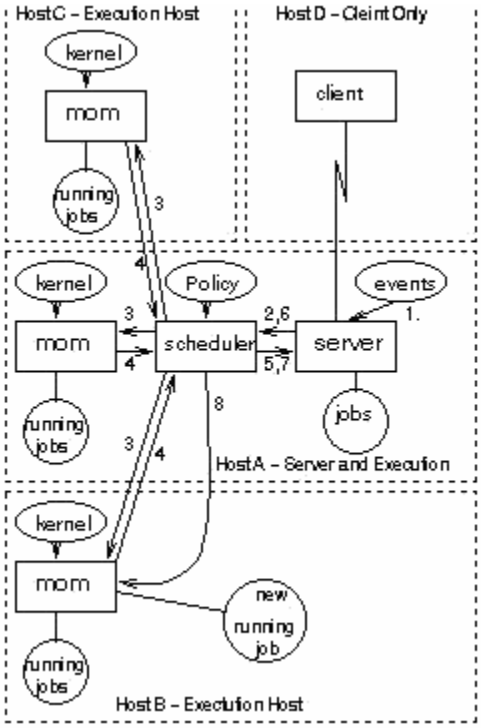
Job Scheduler (b. job scheduler)

The Job Scheduler is another daemon which contains the site's policy controlling which job is run and where and when it is run. Because each site has its own ideas about what is a good or effective policy, PBS allows each site to create its own Scheduler. When run, the Scheduler can communicate with the various Moms to learn about the state of system resources and with the Server to learn about the availability of jobs to execute. The interface to the Server is through the same API as the commands. In fact, the Scheduler just appears as a batch Manager to the Server.

Job Executor (c. resource manager)

The job executor is the daemon which actually places the job into execution. This daemon, *pbs_mom* , is informally called *Mom* as it is the mother of all executing jobs. Mom places a job into execution when it receives a copy of the job from a Server. Mom creates a new session as identical to a user login session as is possible. For example, if the user 's login shell is csh, then Mom creates a session in which .login is run as well as .cshrc. Mom also has the responsibility for returning the job's output to the user when directed to do so by the Server.

G. Functional flow



1. Event tells *server* to initiate a scheduling cycle
2. *Server* sends scheduling command to *scheduler*
3. *Scheduler* requests resource info from *MOM*
4. *MOM* returns requested info
5. *Scheduler* requests job info from *server*
6. *Server* sends job status info to *scheduler*. Scheduler makes policy decision to run job
7. *Scheduler* sends run request to *server*
8. *Server* sends job to *MOM* to run

H. Distribution of control

Central control.

I. Distribution of information services

Distributed Resource Management.

J. Communication architecture

IP based communication.

Interactive jobs and MPMD jobs (more than one executable program) both use sockets (and TCP/IP) to communicate, outside of the job for interactive jobs and between programs in the MPMD case.

K. Interoperability

a. JMS running as a part of another system

- Globus

  Globus is able submit jobs to PBS

c. JMS running concurrently with another system

- NQS

PBS was designed to replace NQS. However, support is provided to aid in the transition from NQS to PBS. The two systems can be run concurrently on the same computer. In addition, the nqs2pbs utility is provided to convert NQS batch job scripts so that they can run under both PBS and NQS.

Major requirements

1. Availability

a. binary code	*Yes*
b. source code	*Yes*
c. documentation	*Yes*
d. roadmap	*Yes*
e. training	*Yes*
f. customer support	*Yes*

Score: *4*

2. Operating systems/platforms

a. submission	**UNIX**
b. execution	**UNIX**
c. control	**UNIX**

PBS is supported on the following computer systems:

Intel- or Alpha-based PC:
- RedHat Linux 5.x, 6.x (Alpha)
- FreeBSD 3.x, 4.0
- NetBSD
- VA Linux
- SGI Linux

Workstations/Servers:
- DEC ALPHA running Digital Unix 4.0D, Tru64 Unix
- HP 9000 running HP-UX 9.x 10.x, 11.x
- IBM RS/6000 running AIX 4.1, 4.2, 4.3
- SGI systems running IRIX 6.1 - 6.5.x
- SUN SPARC running Solaris 2.3 - 2.8

Parallel Supercomputers:
- Cray T3D running UNICOSMK
- Cray T3E running UNICOS/MK2
- IBM SP2 running AIX 3.2, 4.1, 4.2 (with PSSP 2.1), 4.3 (with PSSP 3.1)
- SGI Origin2000 running IRIX 6.4, 6.5.x
- Vector Supercomputers:
- Cray C90 running UNICOS 8, 9, 10
- Cray J90 running UNICOS 8, 9, 10
- Cray SV1 running UNICOS 10
- Fujitsu VPP300 running UXP/V

Score: *3 (No NT)*

3. Batch job support

Score: *4*

4. Interactive job support

Score: *4*

5. Parallel job support

PBS supports parallel programming libraries such as MPI, MPL, PVM, and HPF. Such applications can be scheduled to run within a single multiprocessor systems or across multiple systems.
PBS provides a means by which a parallel job can spawn, monitor and control tasks on remote nodes. See the man page for tm(3). *Unfortunately* , no vendor has made use of this capability though several contributed to its design. Therefore, spawing the tasks of a parallel job fall to the parallel environment itself. PVM provides one means by which a parallel job spawns processes via the pvmd daemon. MPI typically has a vendor dependent method, often using rsh or rexec. All of these means are outside of PBS's control. PBS cannot control or monitor resource usage of the remote tasks, only the ones started by the job on Mother Superior. PBS can only make the list of allocated nodes available to the parallel job and hope that the vendor and the user make use of the list and stay within the allocated nodes. The names of the allocated nodes are place in a file in {PBS_HOME}/aux. The file is owned by root but world readable. The name of the file is passed to the job in the environment variable **PBS_NODEFIL**E. For IBM SP systems, it is also in the variable MP_HOSTFILE. If you are running an open source version of MPI, such as MPICH, then the mpirun command can be modified to check for the PBS environment and use the PBS supplied host file.

Score: *1*

6. Resource requests by users

No sufficient info provided! (see ERS) In System Administrator's Guide they are mentioning:

- CPU time
- Memory
- Disk type
- Network type
- Architecture

Score: *2*

7. Limits on resources by administrators

The Scheduler honors the following attributes/node resources:

Source Object		Attribute/Resource	Comparison
Queue	started	equal true
Queue	queue_type	equal execution
Queue	max_running	ge #jobs running
Queue	max_user_run	ge #jobs running for a user
Queue	max_group_run	ge #jobs running for a group
Job	jobstate	equal Queued
Server	max_running	ge #jobs running
Server	max_user_run	ge #jobs running for a user
Server	max_group_run	ge #jobs running for a group
Server	resources_available	ge resources requested by job
Server	resources_max	ge resources requested
Node	loadave	less than configured limit
Node	arch	equal type requested

Node	host	equal name requested
Node	ncpus	ge number ncpus requested
Node	physmem	ge amount mem requested

a. general *Yes*

Local administrator can specify (in pbs_mom conf. files):
clienthost
restricted
ideal_load
max_load

b. specific users or groups *Yes*

ACLs for the PBS server & queues (able to restrict hosts, users and user groups)

Score: *3*

8. File stage-in and stage-out

PBS provides users with the ability to specify any files that need to be copied onto the execution host before the job runs, and any that need to be copied off after the job completes. The job will be scheduled to run only after the required files have been successfully transferred.

Score: *4*

9. Flexible scheduling

A wide range of scheduling capability is included in the PBS distribution. These can be divided into three types: general-use, system-specific, and instructional.
The General Use scheduler is "FIFO", which (despite its name) implements numerous policies and scheduling algorithms that can be easily tailored to a site's specific needs.
There are also several highly-optimized schedulers for specific systems.
In addition, there are several "sample" schedulers intended to serve as examples for creating a custom scheduler from scratch. Examples are provided for the three scheduler bases: BASL, TCL, and C.

a. several scheduling policies *Yes*

"FIFO" scheduler policies:
round_robin
by_queue
strict_fifo
fair_share
load_balancing
help_starving_jobs
sort_by

b. policies changing in time *No info*

c. scheduling multiple resources simultaneously *Yes*

d. highly configurable scheduling *Yes*

In fact, these are some attributes that can be combined to make new scheduling policies. There are more attributes to specify type of sorting, starving thresholds etc
You can develop your own scheduler.
Highly-optimized schedulers: IBM_SP, SGI_Origin, CRAY T3E, MULTITASK, MSIC-Cluster, DEC-Cluster, UMN-Cluster

Score: *3*

10. Job priorities

Users can specify the priority of their jobs, and defaults can be provided at both the queue and system level.

Score: *2 (no specific info about priority levels; see ERS)*

11. Timesharing

a. processes *Yes*
b. jobs *No*

It is important to remember that the entire node is allocated to one job regardless of the number of processors or the amount of memory in the node.

Score: *2*

12. Impact on owners of computational nodes

Score: *3 (see limitations)*

13. Checkpointing

a. user-level
b. run-time library level *Yes.*
c. OS (kernel) level ***?***

Under Irix 6.5 and later, MPI parallel jobs as well as serial jobs can be checkpointed and restarted on SGI systems provided certain criteria are met. SGI's checkpoint system call cannot checkpoint processes that have open sockets. Therefore it is necessary to tell mpirun to not create or to close an open socket to the array services daemon used to start the parallel processes.

Score: *1*

14. Suspending/resuming/killing jobs

a. user *Yes (qsig)*
b. administrator *Yes*
c. automatic *Yes*

Score: *4*

15. Process migration

a. user *Yes*
b. administrator *Yes*
c. automatic *No info.*

Score: *3*

16. Static load balancing

Score: *4*

17. Dynamic load balancing

No info.

Score: *0*

18. Faul tolerance

c. computational nodes *Yes.*
d. scheduler *No info.*

Score: *2*

19. Job monitoring

Logs, monitoring tools, GUIs.(xpbs)

Score: *4*

20. User interface

CLI & GUI (xpbs)

Score: *4*

21. Published APIs

API available.

Score: *4*

22. Dynamic system reconfiguration

Score: *4*

23. Security

a. Authentication *Yes (privileged port + pbs_iff; Kerberos?)*
b. Authorization *Yes (ACL)*
c. Encryption *?*

Score: *3*

24. Accounting

No info about data points. (see ERS chapter3) PBS currently interfaces with the NASA site wide accounting system, ACCT++, enabling multi-system and multi-site resource accounting.

Score: *4*

25. Scalability

Score: *3*

MAXIMUM TOTAL SCORE = ***75***

Major disadvantages:

no NT support
limited parallel job support
no timesharing of jobs
weak checkpointing support
no dynamic load balancing

Major advantages:

flexible scheduling
GUI to all modules
good authorization mechanism

Reference:

http://pbs.mrj.com/
Portable Batch System OpenPBS Release 2.3 Administrator Guide

A.5 Condor

Background information

A. Authors

University of Wisconsin-Madison (Computer Science Dept.)

B. Support

University of Wisconsin-Madison Graduate School

C. Distribution (commercial/public domain)

Public domain.

D. License

Following are licenses for use of Condor Version 6.0. Academic institutions should agree to the *Academic Use License* for Condor, while all others should agree to the *Internal Use License* for Condor.

System description

E. Objective/ type of the system

a. centralized Job Management System

Condor is a High Throughput Computing environment that can manage very large collections of distributively owned workstations. (It is not uncommon to find problems that require weeks or months of computation to solve. Scientists and engineers engaged in this sort of work need a computing environment that delivers large amounts of computational power over a long period of time. Such an environment is called a High-Throughput Computing (HTC) environment. In contrast, High Performance Computing (HPC) environments deliver a tremendous amount of compute power over a short period of time.) Condor is a software system that runs on a cluster of workstations to harness wasted CPU cycles.

No information regarding the type of clustering (loosely coupled/ tightly coupled). Probably doesn't matter because they are interested in HTC.

F. Major modules

Every machine in a Condor pool can serve a variety of roles. Most machines serve more than one role simultaneously. Certain roles can only be performed by single machines in your pool. The following list describes what these roles are and what resources are required on the machine that is providing that service:

Central Manager

There can be only one central manager for your pool. The machine is the collector of information, and the negotiator between resources and resource requests. These two halves of the central manager's responsibility are performed by separate daemons, so it would be possible to have different machines providing those two services. However, normally they both live on the same machine. This machine plays a very important part in the Condor pool and should be reliable. If this machine crashes, no further matchmaking can be performed within the Condor system (although all current matches remain in effect until they are broken by either party involved in the match). Therefore, choose for central manager a machine that is likely to be online all the time, or at least one that will be rebooted quickly if something goes wrong. The central manager will ideally have a good

network connection to all the machines in your pool, since they all send updates over the network to the central manager. All queries go to the central manager.

Execute

Any machine in your pool (including your Central Manager) can be configured for whether or not it should execute Condor jobs. Obviously, some of your machines will have to serve this function or your pool won't be very useful. Being an execute machine doesn't require many resources at all. About the only resource that might matter is disk space, since if the remote job dumps core, that file is first dumped to the local disk of the execute machine before being sent back to the submit machine for the owner of the job. However, if there isn't much disk space, Condor will simply limit the size of the core file that a remote job will drop. In general the more resources a machine has (swap space, real memory, CPU speed, etc.) the larger the resource requests it can serve. However, if there are requests that don't require many resources, any machine in your pool could serve them.

Submit

Any machine in your pool (including your Central Manager) can be configured for whether or not it should allow Condor jobs to be submitted. The resource requirements for a submit machine are actually much greater than the resource requirements for an execute machine. First of all, every job that you submit that is currently running on a remote machine generates another process on your submit machine. So, if you have lots of jobs running, you will need a fair amount of swap space and/or real memory. In addition all the checkpoint files from your jobs are stored on the local disk of the machine you submit from. Therefore, if your jobs have a large memory image and you submit a lot of them, you will need a lot of disk space to hold these files. This disk space requirement can be somewhat alleviated with a checkpoint server (described below), however the binaries of the jobs you submit are still stored on the submit machine.

Checkpoint Server

One machine in your pool can be configured as a checkpoint server. This is optional, and is not part of the standard Condor binary distribution. The checkpoint server is a centralized machine that stores all the checkpoint files for the jobs submitted in your pool. This machine should have lots of disk space and a good network connection to the rest of your pool, as the traffic can be quite heavy.

Now that you know the various roles a machine can play in a Condor pool, we will describe the actual daemons within Condor that implement these functions:

a. user server

condor schedd This daemon represents resources requests to the Condor pool. Any machine that you wish to allow users to submit jobs from needs to have a *condor schedd* running. When users submit jobs, they go to the schedd, where they are stored in the *job queu*e, which the schedd manages. Various tools to view and manipulate the job queue (such as *condor submi*t, *condor* q, or *condor r*m) all must connect to the schedd to do their work. If the schedd is down on

a given machine, none of these commands will work. The schedd advertises the number of waiting jobs in its job queue and is responsible for claiming available resources to serve those requests. Once a schedd has been matched with a given resource, the schedd spawns a *condor shadow* (described below) to serve that particular request.

b. job scheduler

condor negotiator This daemon is responsible for all the match-making within the Condor system. Periodically, the negotiator begins a *negotiation cycl*e, where it queries the collector for the current state of all the resources in the pool. It contacts each schedd that has waiting resource requests in priority order, and tries to match available resources with those requests. The negotiator is responsible for enforcing user priorities in the system, where the more resources a given user has claimed, the less priority they have to acquire more resources. If a user with a better priority has jobs that are waiting to run, and resources are claimed by a user with a worse priority, the negotiator can preempt that resource and match it with the user with better priority.

condor_schedd

c. resource manager

- resource monitor

condor startd This daemon represents a given resource (namely, a machine capable of running jobs) to the Condor pool. It advertises certain attributes about that resource that are used to match it with pending resource requests. The startd will run on any machine in your pool that you wish to be able to execute jobs. It is responsible for enforcing the policy that resource owners configure which determines under what conditions remote jobs will be started, suspended, resumed, vacated, or killed. When the startd is ready to execute a Condor job, it spawns the *condor starte*r, described below.

condor_schedd

condor shadow This program runs on the machine where a given request was submitted and acts as the resource manager for the request. Jobs that are linked for Condor's standard universe, which perform remote system calls, do so via the *condor shado*w. Any system call performed on the remote execute machine is sent over the network, back to the *condor shadow* which actually performs the system call (such as file I/O) on the submit machine, and the result is sent back over the network to the remote job. In addition, the shadow is responsible for making decisions about the request (such as where checkpoint files should be stored, how certain files should be accessed, etc).

- **job dispatcher** (known also as *job executor*).

condor shadow This program runs on the machine where a given request was submitted and acts as the resource manager for the request. Jobs that are linked for Condor's standard universe, which perform remote system calls, do so via the *condor shado*w. Any system call performed on the remote execute machine is sent over the network, back to the *condor shadow* which actually performs the system call (such as file I/O) on the submit machine, and the result is sent back over the network to the remote job. In addition, the shadow is responsible for making decisions about the request (such as where checkpoint files should be stored, how certain files should be accessed, etc).

d. checkpointing module

condor ckpt server This is the checkpoint server. It services requests to store and retrieve checkpoint files. If your pool is configured to use a checkpoint server but that machine (or the server itself is down) Condor will revert to sending the checkpoint files for a given job back to the submit machine.

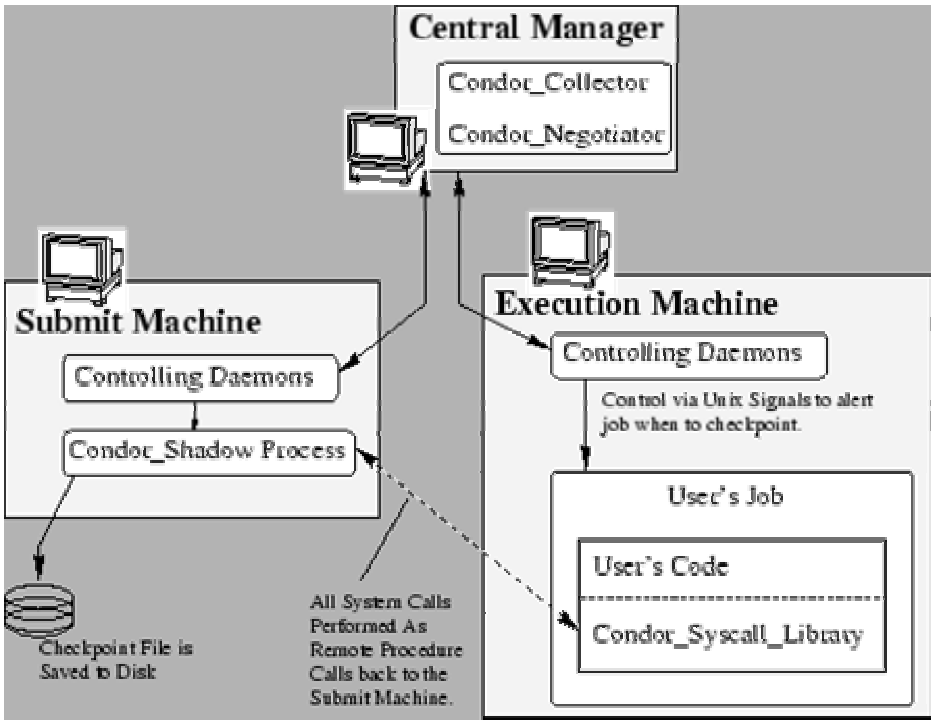
g. fault-detection module:

condor_master This daemon is responsible for keeping all the rest of the Condor daemons running on each machine in your pool. It spawns the other daemons, and periodically

checks to see if there are new binaries installed for any of them. If there are, the master will restart the affected daemons. In addition, if any daemon crashes, the master will send e-mail to the Condor Administrator of your pool and restart the daemon. The *condor master* also supports various administrative commands that let you start, stop or reconfigure daemons remotely. The *condor master* will run on every machine in your Condor pool, regardless of what functions each machine are performing.

condor starter This program is the entity that actually spawns the remote Condor job on a given machine. It sets up the execution environment and monitors the job once it is running. When a job completes, the starter notices this, sends back any status information to the submitting machine, and exits.

G. Functional flow



Pool Architecture

H. Distribution of control

Centralized control (Central Manager)

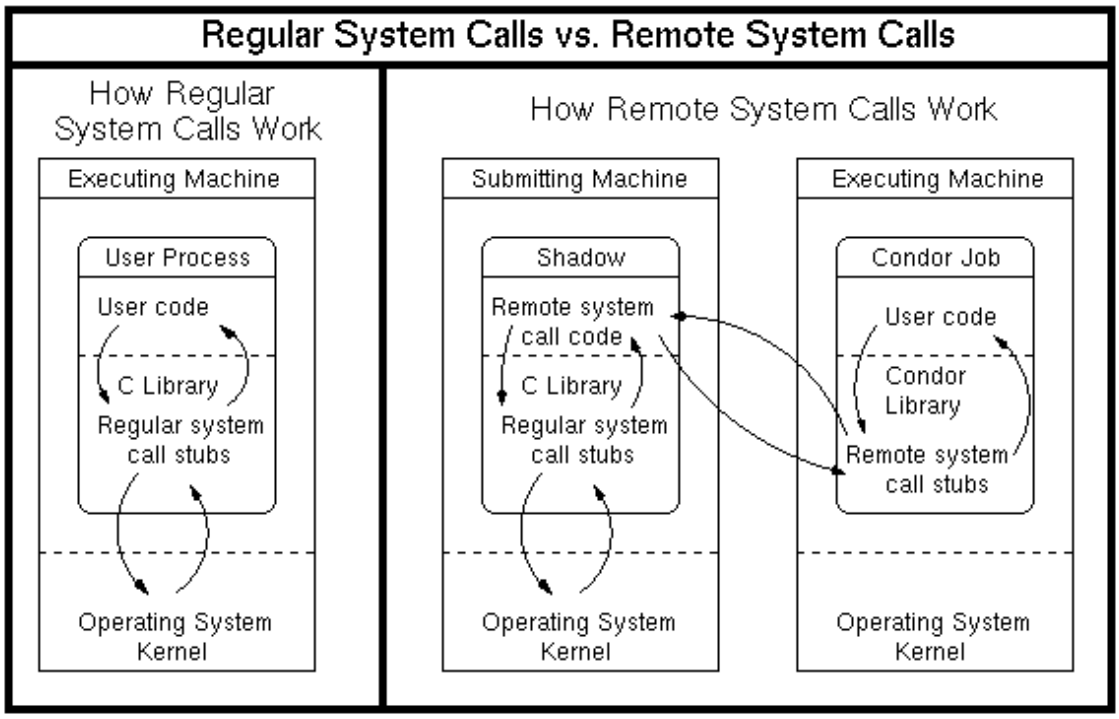
I. Distribution of information services

Central Manager

J. Communication architecture

Condor jobs perform **remote system calls. All system calls are performed as remote procedure calls back to the submitting machine.** In this way, all I/O the job performs is done on the submitting machine, not the executing machine. This is the key to Condor's power in overcoming the problems of distributed ownership. Condor users only have access to the file system on the machine that jobs are submitted from.

Jobs cannot access the file system on the machine where they execute because any system calls that are made to access the file system are simply sent back to the submitting machine and executed there.



A few system calls are allowed to execute on the local machine. These include *sbrk*() and its relatives which are functions that allocate more memory to the job. The only resources on the executing machine a Condor job has access to are the CPU and memory. Of course, the job can only access memory within its own virtual address space, not the memory of any other process. This is insured by the operating system, not Condor.

Condor **does simply not support some system calls**. In particular, the *fork*() system call and its relatives are not supported. These calls create a new process, a copy of the parent process that calls them. This would make it far more complicated to checkpoint, and has some serious security implications as well. By repeatedly forking, a process can fill up the machine with processes, resulting in an operating system crash. If a remote job were allowed to crash a machine by Condor, no one would join a Condor pool. Keeping the owners of machines happy and secure is one of Condor's most important tasks, since without their voluntary participation; Condor would not have access to their resources.

Since *vanilla jobs* are not linked with the Condor library, they are not capable of performing remote system calls. Because of this, they cannot access remote file systems. For a vanilla job to properly function, it must run on a machine with a local file system that contains all the input files it will need, and where it can write its output. Normally, this would only be the submit machine, and any machines that had

a shared file system with it via some sort of network file system like NFS or AFS. Moreover, the job must run on a machine where the user has the same UID as on the submit machine, so that it can access those files properly. In a distributive owned computing environment, these are clearly not necessarily properties of every machine in the pool, though they are hopefully properties of some of them. Condor defines two attributes of every machine in the pool, the UID domain and file system domain. When a vanilla job is submitted, the UID and file system domain of the submit machine are added to the job's requirements. The negotiator will only match this job with a machine with the same UID and file system domain, ensuring that local file system access on the execution machine will be equivalent to file system access on the submit machine.

Socket communication inside parallel jobs (MPI/PVM communication).

K. Interoperability

a. JMS running as a part of another system

The Globus universe allows users to submit Globus jobs through the Condor interface.

b. JMS using another system in place of its module

Condor supports a variety of interactions with Globus software, including running Condor jobs on Globus-managed resources.

Major requirements

1. Availability

a. binary code	*Yes*
b. source code	*Just upon special request.*
c. documentation	*Yes*
d. roadmap	*Yes*
e. training	*Yes*
f. customer support	*For a certain fee.*

Score: ***4***

2. Operating systems/platforms

a. submission	*UNIX/NT*
b. execution	*UNIX/NT*
c. control	*UNIX/NT*

Current functionalities:
1.A single Condor pool can consist of both Windows NT and Unix machines.
2.It does not matter at all if your Central Manager is Unix or NT.
3.Unix machines can submit jobs to run on other Unix or Windows NT machines.
4.Windows NT machines can only submit jobs that will run on Windows NT machines.

Suported architectures:
HPUX10 (for HPUX 10.20)
IRIX6 (for IRIX 6.2, 6.3, or 6.4)
LINUX (for LINUX 2.x kernel systems)
OSF1 (for Digital Unix 4.x)
SOLARIS251
SOLARIS26
NT

Score : *3.5 (the NT version is not a stable version)*

3. **Batch job support**

Score: ***4***

4. **Interactive job support**
No.

Score: ***0***

5. **Parallel job support**
Condor has a PVM submit Universe, which allows the user to submit PVM jobs to the Condor pool. In this section, we will first discuss the differences between running under normal PVM and running PVM under the Condor environment. Then we give some hints on how to write good PVM programs to suit the Condor environment via an example program. In the end, we illustrate how to submit PVM jobs to Condor by examining a sample Condor submit-description file that submits a PVM job.
Also supports MPI jobs.

Score: ***0.5***

6. **Resource requests by users**

No clear specification of the submitting ClassAd attributes. Here are some of them that are mentioned in different places in the Administartor's Guide:

- `Machine type`
- `Architecture`
- `OS`
- `Memory`
- `Virtual memory`
- `CPU time`
- `Wall clock time`
- `Disk`

Probably there are many more.

Score: ***3***

7. Limits on resources by administrators

a. general

Host attributes imposed by the local administrator:

Cpus: Number of CPUs in this machine, i.e. 1 = single CPU machine, 2 = dual CPUs, etc.
CurrentRank: A float which represents this machine owner's affinity for running the Condor job which it is currently hosting. If not currently hosting a Condor job, CurrentRank is -1.0.
Disk: The amount of disk space on this machine available for the job in kbytes (e.g. 23000 = 23 megabytes). Specifically, this is amount of disk space available in the directory specified in the Condor configuration files by the macro EXECUTE, minus any space reserved with the macro RESERVED DISK.
Memory : The amount of RAM in megabytes.
Requirements : A boolean which, when evaluated within the context of the Machine ClassAd and a Job ClassAd, must evaluate to TRUE before Condor will allow the job to use this machine.
UidDomain : a domain name configured by the Condor administrator which describes a cluster of machines which all have the same "passwd" file entries, and therefore all have the same logins.
VirtualMemory : The amount of currently available virtual memory (swap space) expressed in kbytes.

b. specific users or groups

```
Local admin can specify:
RsrchGrp = { "raman", "miron", "solomon", "jbasney" };
Friends = { "tannenba", "wright" };
Untrusted = { "rival", "riffraff" };
Rank = member(other.Owner, RsrchGroup) ? 10 : 0 + member(other.Owner
         , Friends) ?1 :0;
Constraint = !member(other.Owner, Untrusted) && (Rank == 10 ? true
   : Rank > 0 ? (LoadAvg < 0.3 && KbdIdle > 15*60)
   : (DayTime < 8*60*60 || DayTime > 18*60*60)))
```

Score: *4*

8. File stage-in and stage-out

Current limitations:

- Transfer of subdirectories is not performed. When starting your job, Condor will create a temporary working directory on the execute machine and place your executable and all input files into this directory. Condor will then start your job with this directory as the current working directory. When your job completes, any files created in this temporary working directory are transferred back to the submit machine. However, if the job creates any subdirectories, files in those subdirectories are not transferred back. Similarly, only filenames, not directory names, can be specified with the **transfer input files** submit-description file parameter.
- Running out of disk space on the submit machine is not handled as gracefully as it should be.
- By default, any files created or modified by the job are automatically sent back to the submit machine. However, if the job deleted any files in its temporary working directory, they currently are not deleted back on the submit machine. This could cause problems if **transfer files** is set to *ALWAYS* and the job uses the presence of a file as a lock file. Note there is no problem if **transfer files** is set to the default, which is *ONEXIT*.

Score: ***3***

9. Flexible scheduling

Scheduling policies are definable. Condor is shipped with one default policy. There are some attributes you can define in order to define a policy for you Condor pool.

a. several scheduling policies ***No***

b. policies changing in time ***No***

c. scheduling multiple resources simulataneously *No info. Probably yes.*
d. highly configurable scheduling *Yes*

Score: *2*

10. Job priorities

There are job priorities and user priorities.
Job priorities are ranging from –20 to +20.
Job priorities can be changed by user or by admin.

Score: *4*

11. Timesharing

a. processes *Yes*
b. jobs *No info.*

Score: *2*

12. Impact on owners of computational nodes

A machine may be configured to prefer certain jobs to others using the `RANK` expression. It is an expression, like any other in a machine ClassAd. It can reference any attribute found in either the machine ClassAd or a request ad (normally, in fact, it references things in the request ad). The most common use of this expression is likely to configure a machine to prefer to run jobs from the owner of that machine, or by extension, a group of machines to prefer jobs from the owners of those machines.

Score: *4*

13. Checkpointing

a. user-level *Yes (API to the library)*
b. run-time library level *Yes*
c. OS (kernel) level *No*

Limitations on Jobs which can Checkpointed

Although Condor can schedule and run any type of process, Condor does have some limitations on jobs that it can transparently checkpoint and migrate:
1. Multi-process jobs are not allowed. This includes system calls such as `fork()`, `exec()`, and `system()`.
2. Interprocess communication is not allowed. This includes pipes, semaphores, and shared memory.
3. Network communication must be brief. A job *may* make network connections using system calls such as `socket()`, but a network connection left open for long periods will delay checkpointing and migration.
4. Sending or receiving the SIGUSR2 or SIGTSTP signals is not allowed. Condor reserves these signals for its own use. Sending or receiving all other signals *is* allowed.
5. Alarms, timers, and sleeping are not allowed. This includes system calls such as `alarm()`, `getitimer()`, and `sleep()`.
6. Multiple kernel-level threads are not allowed. However, multiple user-level threads *are* allowed.
7. Memory mapped files are not allowed. This includes system calls such as `mmap()`and `munmap()`.
8. File locks are allowed, but not retained between checkpoints.

9. All files must be opened read-only or write-only. A file opened for both reading and writing will cause trouble if a job must be rolled back to an old checkpoint image. For compatibility reasons, a file opened for both reading and writing will result in a warning but not an error.
10. A fair amount of disk space must be available on the submitting machine for storing a job's checkpoint images. A checkpoint image is approximately equal to the virtual memory consumed by a job while it runs. If disk space is short, a special *checkpoint server* can be designated for storing all the checkpoint images for a pool.
11. On Digital Unix (OSF/1), HP-UX, and Linux, your job must be statically linked. Dynamic linking is allowed on all other platforms.
Note: these limitations *only* apply to jobs that Condor has been asked to transparently checkpoint. If job checkpointing is not desired, the limitations above do not apply.

Score: *2*

14. Suspending/resuming/killing jobs

a. user
b. administrator
c. automatic

Score: *4*

15. Process migration

a. user ***Probably yes***
b. administrator ***Probably yes***
c. automatic ***Yes (just for checkpointed jobs)***

Score: *2 (has the same limitations as the checkpointing mech.)*

16. Static load balancing

CPU/Memory consideration. No detail how it is done.

Score: ***4***

17. Dynamic load balancing

No info. Probably not.

Score: *0*

18. Faul tolerance

a. computational nodes **Yes *(checkpointing + migration)***
b. scheduler

If the central manager crashes, jobs that are already running will continue to run unaffected. Queued jobs will remain in the queue unharmed, but cannot begin running

until the central manager is restarted and begins matchmaking again. Nothing special needs to be done after the central manager is brought back online.

Score: 3

19. Job monitoring
Logs + GUI

Score: *4*

20. User interface
CLI + GUI

Score: *4*

21. Published APIs

Score: *4*

22. Dynamic system reconfiguration

Score: *4*

23. Security

a.	**authentication**	***X.509 certificates (SSL)***
b.	**authorization**	***different access levels***
c.	**encryption**	***No info.***

Score: *3*

24. Accounting

By default, Condor will send you an email message when your job completes. You can modify this behavior with the *condor submit* "notification" command. The message will include the exit status of your job (i.e., the argument your job passed to the exit system call when it completed) or notification that your job was killed by a signal. It will also include the following statistics (as appropriate) about your job:
Submitted at: when the job was submitted with *condor submit*
Completed at: when the job completed
Real Time: elapsed time between when the job was submitted and when it completed (days: hours:minutes:seconds)
Run Time: total time the job was running (i.e., real time minus queueing time)
Committed Time: total run time that contributed to job completion (i.e., run time minus the run time that was lost because the job was evicted without performing a checkpoint)
Remote User Time: total amount of committed time the job spent executing in user mode
Remote System Time: total amount of committed time the job spent executing in system mode
Total Remote Time: total committed CPU time for the job

Local User Time: total amount of time this job's *condor shadow* (remote system call server) spent executing in user mode
Local System Time: total amount of time this job's *condor shadow* spent executing in system mode
Total Local Time: total CPU usage for this job's *condor shadow*
Leveraging Factor: the ratio of total remote time to total system time (a factor below 1.0 indicates that the job ran inefficiently, spending more CPU time performing remote system calls than actually executing on the remote machine)
Virtual Image Size: memory size of the job, computed when the job checkpoints
Checkpoints written: number of successful checkpoints performed by the job
Checkpoint restarts: number of times the job successfully restarted from a checkpoint
Network: total network usage by the job for checkpointing and remote system calls
Buffer Configuration: configuration of remote system call I/O buffers
Total I/O: total file I/O detected by the remote system call library
I/O by File: I/O statistics per file produced by the remote system call library
Remote System Calls: listing of all remote system calls performed (both Condor-specific and Unix system calls) with a count of the number of times each was performed

There for it must have some accounting capabilities.

Score: *4*

25. Scalability

Score: *3*

TOTAL SCORE = **75**

Major disadvantages:

No interactive job support
Limited parallel job support
No timesharing of jobs
Limited checkpointing
No dynamic load balancing

Major advantages:

Minimal impact on owners of computational nodes
GUI to all modules
Strong authentication and authorization

Reference:

http://www.cs.wisc.edu/condor/
Condor Version 6.1.17 Manual

A.6 RES

Background information

A. Authors

William W. Carlson
Supercomputing Research Center
Institute for Defense Analyses

B. Support

Unknown

C. Distribution (commercial/public domain)

Commercial (restricted) (?)

D. License

Unknown

System description

E. Type of system

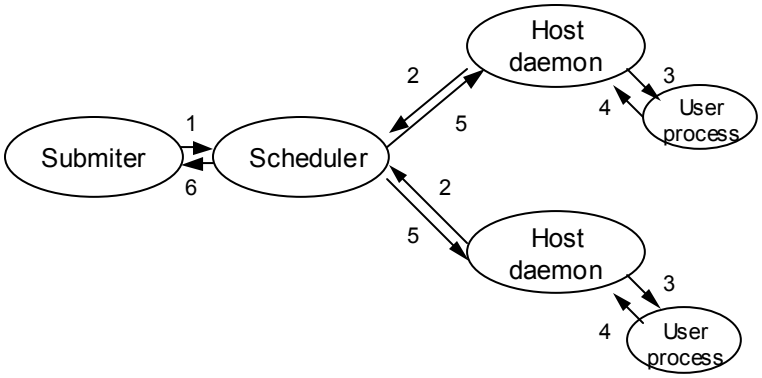
Centralized job management

RES is a simple system, which allows users to effectively distribute computation among network resources that would otherwise be unused. The defining characteristics of such computation is, that it be partitionable into relatively small chunks of work that do not require significant communication between them.

F. Major modules

- Job submitter (*user server*)
- Central scheduler (*job scheduler* combined with *resource manager*)
  - Arrival queue
  - Waiting queue
  - Running queue
  - Done queue
- Execution host daemon (*job dispatcher*)

G. Functional flow



1 – User submits a job
2 – Job is sent to host daemon
3 – Host daemon executes job
4 – Jobs return results to host daemon
5 – Host daemon sends results do scheduler
6 – User retrieves results from scheduler

H. Distribution of control

Central scheduler, all users have administrative privileges

I. Distribution of information services

The host daemon reports periodically various parameters to central scheduler with which the scheduler will use to make its decisions. To collect this information daemon contacts local statistics daemon (rstatd). The reporting period varies based on the status of the daemon, when it last executed work, and the state of central scheduler. The strategy requires that frequent reports (1 per a minute) be made when work is active or has been active recently. If the system has been unused for a long period of time, these reports are not required to come so often and the system backs off to less frequent periods (4 per an hour). If the daemon detects the scheduler is unavailable, it will delay reports for up to an hour. Daemon reports following parameters: CPU utilization, IO operations counts, network activity, idle time, physical and virtual memory availability, NFS activity.

J. Communication

a. Protocols

IP based; uses RPC for communication

b. Shared file systems

NFS is used for share files among workstations

K. Interoperability

c. JMS running as a part of another system

Can accept jobs submitted from other systems

d. JMS using another system in place of its module

Not possible

e. JMS running concurrently with another system

Not possible

Major requirements

1. **Availability and quality of:**
   a. Binary code — available
   b. Source code — available
   c. Documentation — Good (technical report)
   d. Roadmap — Improved system error recovery, distributed global scheduler, site-specific job descriptions, support for user-level check pointing, higher level of security
   e. Training — unknown
   f. Customer support — unknown

   Score: *2*

2. **Operating systems/platforms**
   a. Submission — *NIX
   b. Execution — *NIX
   c. Control — *NIX

   Tested on:

   - Sun 3Sun 4SGIBM RS/6000
     Should work on any UNIX-like system

   Score: *2*

3. **Batch job support**
   Score: *4*

4. **Interactive job support**
   Score: *0*

5. **Parallel job support**
   Score: *0*

6. **Resource requests by users**
   Yes (Execution time, memory size, resident set size, I/O operations count, architecture)
   Score: *3*

7. **Limits on resources by administrators**
   a. General — Restrictions on queues
   b. Specific users or groups — No
   Score: *2*

8. **File stage-in and stage-out**
   Score: *0*

9. **Flexible scheduling**
   a. Several scheduling policies — No

b. Policies changing in time	No
c. Scheduling multiple resources in time	Yes
d. Highly configurable scheduling	No
Score:	*1*

10. Job priorities

a. User assigned	No
b. System assigned	Yes
Score:	*2*

11. **Timesharing**

a. Processes	No
b. Jobs	No
Score:	*0*

12. Impact on owners of computational nodes

a. Configurability	by turning on and off host daemon
b. Influence	Small
Score:	*4*

13. Check pointing

a. User-level	Not aware
b. Run-time library level	No
c. OS (kernel) level	Not aware
Score:	*1*

14. **Suspending/resuming/killing jobs**

a. User	Yes (killing)
b. Administrator	Yes (killing)
c. Automatic	Yes
Score:	*3*

15. **Process migration**

a. User	No
b. Administrator	No
c. Automatic	No
Score:	*0*

16. Static load balancing

Yes

Score: *4*

17. Dynamic load balancing

No

Score: *0*

18. Fault tolerance

a. Computational nodes	Yes (job will be rescheduled)
b. Scheduler	No
Score:	*2*

19. Job monitoring

Runtime statistics, on the status of jobs and queues

Score: *3*

20. User interface

CLI

Score: *3*

21. Published APIs

Score: *4*

22. Dynamic system reconfiguration

Adding/removing computational nodes, changing some parameters of scheduler

Score: *3*

23. **Security**

a. Authentication	Yes
b. Authorization	No
c. Encryption	No
Score:	*1*

24. Accounting

Score: *0*

25. Scalability

Score: *4*

MAXIMUM TOTAL SCORE = **48**

Advantages

- Minimal impact on owners of computational nodes

Disadvantages

- No support for Windows NT nor Linux
- No interactive jobs support
- No parallel jobs support
- No stage-in nor stage-out
- No time sharing support
- No check pointing
- No process migration
- Limited fault tolerance

- No authorization mechanisms
- No accounting capabilities

References:

William W. Carlson, "*RES: A simple system for distributed computing*", Technical Report SRC-TR-92-067, Supercomputing Research Center

A.7 MOSIX

Background information

A. Authors
The Hebrew University of Jerusalem, Institute of Computer Science

B. **Support**
The Hebrew University of Jerusalem, Institute of Computer Science

C. Distribution (commercial/public domain)
Free

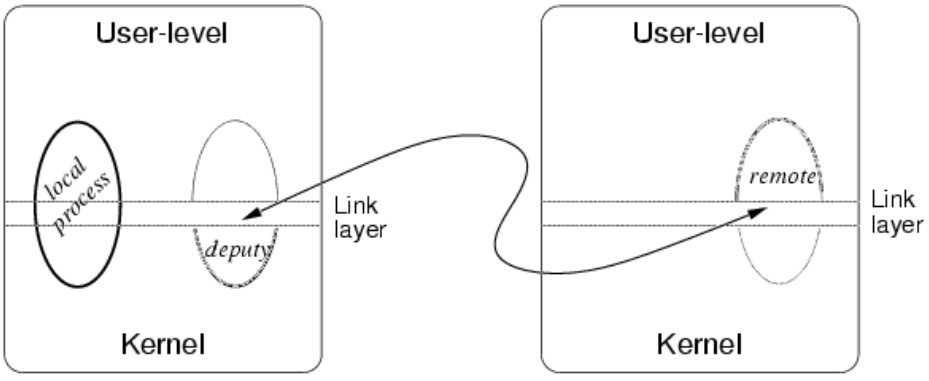
D. License
GNU

System description

E. Type of the system
Distributed operating system. A cluster computing enhancement of Linux that supports preemptive process migration. It consists of kernel-level, adaptive resource sharing algorithms that are geared for high-performance, overhead-free scalability and ease-of-use of a scalable computing cluster.
Appropriate for tightly coupled clusters of workstations.

F. Major modules
Scheduler (with load balancing)
Resource manager
Process queues
Network communication module
Authentication modules
User interface

G. Functional flow



Mosix supports preemptive (completely transparent) process migration (PPM). After a migration, a process continues to interact with its environment regardless of its location. To implement the PPM, the migrating process is divided into two contexts: the user context - that can be migrated, and the system context - that is user host node (UHN) dependent, and may not be migrated.

The user context, called the remote, contains the program code, stack, data, memory-maps and registers of the process. The remote encapsulates the process when it is running in the user level. The system context, called the deputy, contains description of the resources, which the process is attached to, and a kernel-stack for the execution of system code on behalf of the process. The deputy encapsulates the process when it is running in the kernel. It holds the site dependent part of the system context of the process, hence it must remain in the UHN of the process. While the process can migrate many times between different nodes, the deputy is never migrated.
The interface between the user-context and the system context is well defined. Therefore it is possible to intercept every interaction between these contexts, and forward this interaction across the network. This is implemented at the link layer, with a special communication channel for interaction.
In the execution of a process in Mosix, location transparency is achieved by forwarding site dependent system calls to the deputy at the UHN. System calls are a synchronous form of interaction between the two process contexts. All system calls, that are executed by the process, are intercepted by the remote site's link layer. If the system call is site independent it is executed by the remote locally (at the remote site). Otherwise, the system call is forwarded to the deputy, which executes the system call on behalf of the process in the UHN. The deputy returns the result(s) back to the remote site, which then continues to execute the user's code.
Other forms of interaction between the two process contexts are signal delivery and process wakeup events, e.g. when network data arrives. These events require that the deputy asynchronously locate and interact with the remote.
This location requirement is met by the communication channel between them. In a typical scenario, the kernel at the UHN informs the deputy of the event. The deputy checks whether any action needs to be taken, and if so, informs the remote. The remote monitors the communication channel for reports of asynchronous events, e.g., signals, just before resuming user-level execution. We note that this approach is robust, and is not affected even by major modifications of the kernel. It relies on almost no machine dependent features of the kernel, and thus does not hinder porting to different architectures.

H. Distribution of control

Distributed control

I. Distribution of information services

Nodes send periodically information about resource allocation to randomly selected nodes

J. Communication architecture

a. Protocols:

TCP/IP, fast networks with low latency

b. Shared file systems

Support for many types of shared file systems implemented in the kernel of the system (NFS, AFS, SMB), implemented new distributed file system implementation MFS (Mosix File System) with DFSA (Direct File System Access) improving the performance of access to remote files.

K. Interoperability

a. JMS running as a part of another

Any JMS can submit a job to the cluster of workstations running Mosix

b. JMS using another system in place of its module

No

c. JMS running concurrently with another system

No

Major requirements

1. **Availability**
   - a. Binary code: Yes
   - b. Source code: Yes
   - c. Documentation: Good (User documentation, research papers)
   - d. Roadmap: High availability, scalable web servers, cluster installation, network RAM, migratable sockets
   - e. Training: No
   - f. Customer support: Yes

   Score: *3*

2. **Operating systems/platforms**
   - a. Submission: Linux/BSD
   - b. Execution: Linux/BSD
   - c. Control: Linux/BSD

   Score: *1*

3. **Batch job support**

   Score: *2*

4. **Interactive job support**

   Score: *4*

5. **Parallel job support**

   Score: *4*

6. **Resource requests by users**

   Score: *0*

7. **Limits on resources by administrators**
   - a. General: Yes
   - b. Specific users or groups: Yes

   Score: *1*

8. **File stage-in and stage-out**

   Score: *0*

9. **Flexible scheduling**
   - a. Several scheduling policies: Yes
   - b. Policies changing in time: No
   - c. Scheduling multiple resources simultaneously: Yes
   - d. Highly configurable scheduling: No

   Score: *1*

10. Job priorities

From –20 to 20

Score: *0*

11. **Timesharing**

a. Processes	Yes
b. Jobs	Yes

Score: *4*

12. Impact on owners of computational nodes

a. Configurability

Users can specify whether their workstation can be a part of a cluster and whether can execute other users jobs.

b. Influence

Average

Score: *1*

13. Check pointing

a. User-level	No
b. Run-time library level	No
c. OS (kernel) level	No

Score: *0*

14. Suspending/resuming/killing jobs

a. User	Yes
b. Administrator	Yes
c. Automatic	Yes

Score: *3*

15. **Process migration**

a. User	Yes/No
b. inistrator	No
c. automatic	Yes

Score: *2*

16. Static load balancing

Score: *4*

17. Dynamic load balancing

Score: *4*

18. Fault tolerance

a. Computational nodes	Yes
b. Scheduler	Yes

Score: *2*

19. Job monitoring
Score: *4*

20. User interface
Score: *4*

21. Published APIs
Score: *4*

22. Dynamic system reconfiguration
Score: *4*

23. Security

a. Authentication	Yes
b. Authorization	Yes
c. Encryption	No

Score: *3*

24. Accounting
Score: *0*

25. Scalability
Score: *2*

MAXIMUM TOTAL SCORE = **57**

Advantages

- Transparent process migration
- Interactive job support
- Integration with distributed file systems
- Strong authentication and authorization

Disadvantages

- Not a JMS
- Supports only Linux and BSD
- Supports only tightly coupled clusters of homogenous workstations
- Impact on owners of computational nodes
- No check pointing

References:

Homepage: `http://www.mosix.org/`

Barak A. and La'adan O., *"The MOSIX Multicomputer Operating System for High Performance Cluster Computing"*, Journal of Future Generation Computer Systems, Vol. 13, No. 4-5, pp. 361-372, March 1998.
`http://www.mosix.cs.huji.ac.il/ftps/mosixhpcc.ps.gz`

Barak A., La'adan O. and Shiloh A., "*Scalable Cluster Computing with MOSIX for LINUX*", Proceedings Linux Expo '99,pp. 95-100, Raleigh, N.C., May 1999.
http://www.mosix.cs.huji.ac.il/ftps/mosix4linux.ps.gz

Amar L., Barak A., Eizenberg A. and Shiloh A., "*The MOSIX Scalable Cluster File Systems for LINUX*", July 2000.
http://www.mosix.cs.huji.ac.il/ftps/mfs.ps.gz

Slide presentation At Linux EXPO Paris/Linuxworld, Feb. 2, 2001.
http://www.mosix.cs.huji.ac.il/slides/Paris/paris.htm

Mosix reference manual
http://www.mosix.cs.huji.ac.il/ftps/mosix.man.html

A.8 GLOBUS

Background information

A. Authors

Argonne National Laboratory
University of Southern California Information Sciences Institute

B. Support

Argonne National Laboratory
The University of Chicago,
University of Southern California Information Sciences Institute
High Performance Computing Laboratory Northern Illinois University
National Center for Supercomputing Applications at the University of Illinois Urbana-Champaign
National Aeronautics and Space Administration

C. Distribution

Public domain.

D. License

Globus Toolkit Public License (GTPL)

System Description

E. Objective/ type of the system

a. distributed Job Management System without a central job scheduler

The Globus Project is developing the fundamental technology that is needed to build computational grids. Research of the Globus Project targets technical challenges in, for example, communication, scheduling, security, information, data access, and fault detection. Development of the Globus Project focuses on the Globus Toolkit, an integrated set of tools and software that facilitate the creation of applications that can exploit the advanced capabilities of the grid and the creation of production or prototype grid infrastructures, using a **combination of the Globus Toolkit and other technologies**.

Globus is designed to offer features such as uniform access to distributed resources with diverse scheduling mechanisms; information service for resource publication, discovery, and selection; API and command-line tools for remote file management, staging of executables and data; and enhanced performance through multiple communication protocols.

F. Major modules

The components of the Globus Toolkit can be used either independently or together to develop useful grid applications and programming tools.

Globus Resource Allocation Manager (GRAM)
(c. Resource manager + a. User server)

The GRAM provides resource allocation and process creation, monitoring, and management services. GRAM implementations map requests expressed in a Resource Specification Language (RSL) into commands to local schedulers and computers. GRAM is one of the Globus Toolkit components that may be used independently. The user interface to GRAM is the *gatekeepe*r.

Grid Security Infrastructure (GSI)
(f. security module)

The GSI provides a single sign-on authentication service, with support for local control over access rights and mapping from global to local user identities. GSI support for hardware tokens increases credential security. GSI may be used independently and, in fact, has been integrated into numerous programs that are independent of the rest of the Globus Toolkit.

Metacomputing Directory Service (MDS)
(c. Resource manager)
The MDS is an integrated information service distributed across Globus-enabled resources that provides information about the state of the Globus grid infrastructure. The service is based on the Lightweight Directory Access Protocol (LDAP).

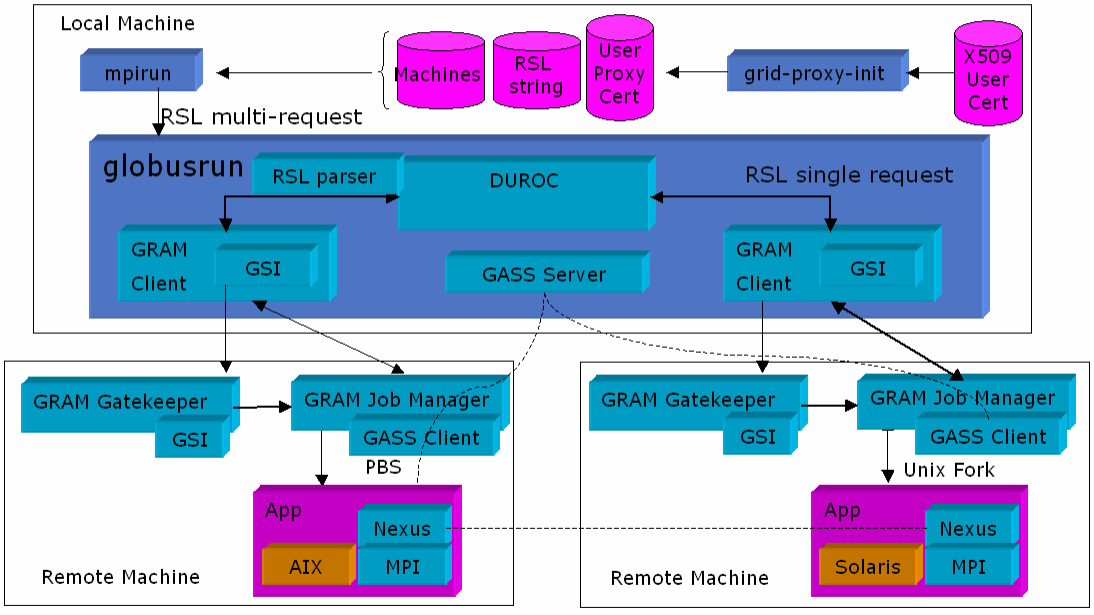
Global Access to Secondary Storage (GASS)
The GASS service implements a variety of automatic and programmer-managed data movement and data access strategies, enabling programs running at remote locations to read and write local data.

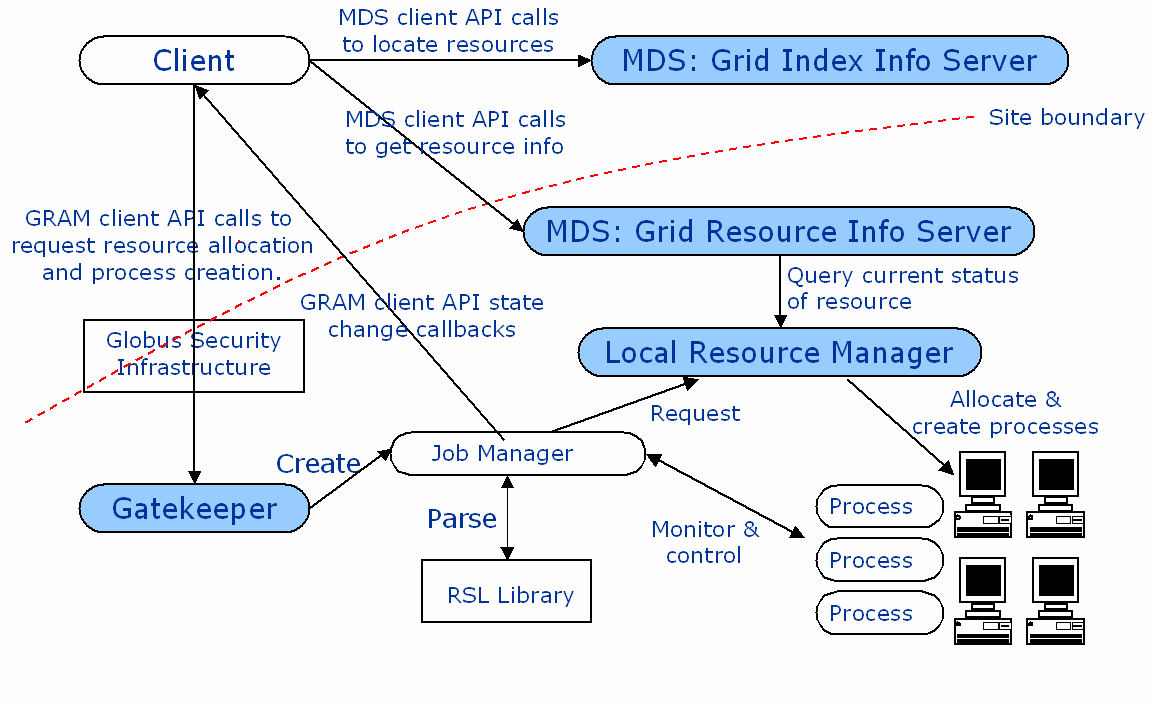
Globus Toolkit I/O
Globus Toolkit I/O provides an easy-to-use interface to TCP, UDP, and file I/O. It supports both synchronous and asynchronous interfaces, multithreading, and integrated GSI security.

Nexus
Nexus provides communication services for heterogeneous environments, supporting multimethod communication, multithreading, and single-sided operations.

Heartbeat Monitor (HBM)
(g. fault-detection module)
The HBM allows you or your users to detect failure of the Globus Toolkit components or application processes.

G. Functional flow
In the figure below, we can observe a typical mpirun call submited by an authenticated user.





We must be aware that GLOBUS is designed like a toolkit that can be used to build a real JMS. It is NOT a JMS. Does not have a built-in scheduler, but it can be used in conjunction with another scheduling system as mentioned below.

H. Distribution of control

Distributed control.

I. Distribution of information services

Based on LDAP. GRIS servers are reporting to a central GIIS server. The Globus Toolkit ships with one GIIS. But there should be more than one GIIS server in a large cluster.

J. Communication architecture

See GASS and NEXUS modules.
Uses http, shttp, ftp, LDAP.
It is recommended to install GLOBUS on a shared file system, but they don't say anything about functions integrated from a DFS.

K. Interoperability

b. JMS using another system in place of its module

Globus is designed to offer uniform access to distributed resources with diverse scheduling mechanisms. The following scheduling interfaces are supported by the resource management architecture of the Globus Toolkit:

- Unix fork (the default scheduler)
- POE
- Condor
- Easy-LL (easymcs)
- NQE (on Cray T3E)
- Prun

- Loadleveler
- LSF
- PBS
- GLUnix
- Pexec

Globus can submit jobs to all the schedulers mentioned above.

Major requirements

1. **Availability**
   - **a. binary code** *Yes*
   - **b. source code** *Yes*
   - **c. documentation** *Yes*
   - **d. roadmap** *Yes*
     Data transfer (GSI-FTP)
     NT
     Replica Management
   - **e. training** *No*
   - **f. customer support** *No*

   Score: ***3***

2. **Operating systems/platforms**
   - **a. submission** *UNIX*
   - **b. execution** *UNIX*
   - **c. control** *UNIX*

   Tested platforms:
   AIX, Digital Unix, FreeBSD, HPUX, IRIX, Linux (Intel-based), Solaris, UNICOS/mk.

   Score: *3 (No NT version)*

3. **Batch job support**
   globus-job-submit
   Score: ***4***

4. **Interactive job support**
   globus-job-run
   Score: ***4***

5. **Parallel job support**
   Works with MPICH-G and Unix fork.

   Score: ***1***

6. **Resource requests by users**
   RSL = Resource Specification Language (very powerful tool)

No clear specification of the requested resource attributes. Here are some of them that are mentioned in different places in the User Tutorials:

- Machine type
- Architecture
- OS
- Memory
- System time
- Disk space
- Swap space
- Network type
- Executable name
- Number of instances (of the job)
- Number of processors
- MFLOPS (CPU time)

Probably there are many more.

Score: ***4***

7. **Limits on resources by administrators**
   a. **general**
   b. **specific users or groups**

Probably there is a large quantity of attributes (limits) that can be specified by administrators. They are not clearly specified.

Score: *2*

8. **File stage-in and stage-out**
   Acomplished by the GASS module.

Score: ***4***

9. **Flexible scheduling**
   a. several scheduling policies ***No***
   b. policies changing in time ***No***
   c. scheduling multiple resources simultaneously ***Yes (DUROC)***
   d. highly configurable scheduling ***Yes***

The following scheduling interfaces are supported by the resource management architecture of the Globus Toolkit:

- Unix fork (the default scheduler)
- POE
- Condor
- Easy-LL (easymcs)
- NQE (on Cray T3E)
- Prun
- Loadleveler
- LSF
- PBS
- GLUnix

Score: ***2***

10. Job priorities

No info.

Score: ***0***

11. Timesharing

d. processes ***Yes***

e. jobs ***Yes***

Score: ***4***

12. Impact on owners of computational nodes

NO info.

Score: *0*

13. Checkpointing

a. user-level ***No***

b. run-time library level ***No***

c. OS (kernel) level ***No***

Score: *0*

14. Suspending/resuming/killing jobs

a. user ***Yes***

b. administrator ***No info.***

c. automatic ***No info.***

Score: *1*

15. Process migration

c. user ***No info.***

b. administrator ***No info.***

c. automatic ***No info.***

Score: *0*

16. Static load balancing

You can implement your own resource broker to do load balancing.

Score: ***1***

17. Dynamic load balancing

No info. Probably not.

Score: *0*

18. Faul tolerance

a. computational nodes **Yes *(checkpointing + migration)***

b. scheduler

Just fault detection. (HBM)

Score: *1*

19. Job monitoring

HBM produces log files and has a GUI .

Score: *4*

20. User interface

CLI + GUI

Score: *4*

21. Published APIs

Score: *4*

22. Dynamic system reconfiguration

Score: *4*

23. Security

a. authentication *X.509 certificates (SSL) + KERBEROS*

b. authorization ***user map files***

c. encryption ***DES***

Score: *4*

24. Accounting

Resource usage accounting is presumed to be handled locally by each site. The Globus Toolkit does not change any existing local accounting mechanisms. Globus Toolkit jobs run under the user account as specified in the grid-mapfile. Therefore, a user is required to already have a conventional Unix account on the host to which the job is submitted.

Score: *0*

25. Scalability

Runs on GUSTO test-bed (17 sites, 330 computers, 3600 procs).

Score: *4*

TOTAL SCORE = **58**

Major disadvantages:
No central scheduling
No Windows NT support
Limited parallel job support
No checkpointing
No process migration
Limited fault tolerance
No accounting capabilities

Major advantages:
High portability
High interoperability
Strong authentication, authorization, and encryption
Very scalable
Stage-in and stage-out

Reference:

http://www.globus.org/
Globus Toolkit 1.1.3 System Administration Guide *December 2000*
Globus Tutorials

A.9 LEGION

Background information

A. Authors

University of Virginia Computer Science Department

B. Support

University of Virginia Computer Science Department

C. Distribution

Public domain.

D. License

Licensing agreement.

System description

E. Objective/ type of the system

b. distributed Job Management System without a central job scheduler

Object-based distributed job management system intended to support the construction of wide-area virtual computers, or metasystems. Grid oriented. It has no centralized scheduling mechanism but you can associate a scheduler for every particular application.

F. Major modules

In the allocation of resources for a specific task there are three steps: decision, enactment, and monitoring. In the decision stage, the task's characteristics, requirements, and run-time behavior, the resource's properties and policies, and users' preferences must all be considered. Legion provides an information infrastructure and a resource negotiation framework for this stage. The allocation decision includes target hosts and storage space with assorted host objects and vaults. At the enactment stage, the allocation coordinator sends an activation request, including the desired host-vault mapping for the object, to the class object that will carry out the request. The class object checks that the placement is acceptable and then coordinates with the host object and vault to create and start the new object. The monitoring stage ensures that the new object is operating correctly and that it is using the correct allocation of resources. An object-mandatory interface includes functions to establish and remove triggers that will monitor object status.

There are three special objects involved in Legion resource management: the ***collection***, the ***scheduler***, and the ***enactor***.

The **collection** collects resource information, constantly monitoring the system's host and vault objects to determine which resources are in use and which are available for what kind of tasks.

The **scheduler** determines possible resource use schedules for specific tasks and makes policy decisions about placing those tasks.

The **enactor** negotiates with resources to carry out those schedules and acquires reservation tokens from successful negotiations.

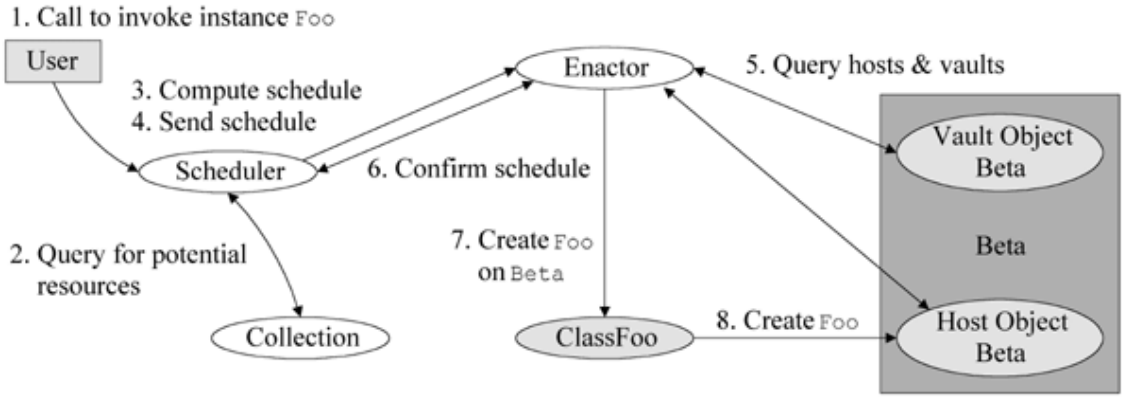
a. User server (known also as *job server*) = *scheduler*

c. Resource manager

- **resource monitor** = *collection*
- **job dispatcher** = *enactor*

G. Functional flow

If a user wants ClassFoo to start instance Foo on another host, the user should send a call (Figure 20, step 1) to the basic Legion scheduler (ClassFoo can also have an associated external scheduler so that a user could call the class and the class would then call its scheduler.). The scheduler then consults the collection to determine what resources are appropriate and available for Foo (step 2) and builds a sample schedule or series of schedules (step 3). It then sends a sample schedule to the enactor (step 4). The enactor contacts each resource on the schedule and requests an allocation of time (step 5). Once it has contacted each resource and reserved the necessary time, it confirms the schedule with the scheduler (step 6), and then contacts ClassFoo and tells it to begin Foo on the appropriate resources (step 7). ClassFoo contacts the resource(s) and sends the order to create Foo (step 8).



H. Distribution of control

Distributed control.

I. Distribution of information services

Distributed information service. (*CONTEXT SPACE = distributed collection of objects*)

J. Communication

a. protocols

Uses its own data transfer protocols:

legion-read-udp, legion-write-udp
lio-read-udp, lio-write-udp
lnfs-read-udp, lnfs-write-udp
legion-read-tcp, legion-write-tcp
lio-read-tcp, lio-write-tcp
lnfs-read-tcp, lnfs-write-tcp

b. shared file systems

Good performances compared with ftp and Globus GASS.
There is a modified, Legion-aware, NFS daemon, `lnfsd` that receives NFS requests from the kernel and translates these into the appropriate Legion method invocations. Upon receiving the results, it packages them in a form digestible by the NFS client. The file system is mounted like any NFS file system to maintain this level of security by the fixed interface between the NFS (kernel) client and `lnfsd`. In this way *Legion* provides a virtual file system that spans all the machines in a Legion system. Input and output files can be seen by all the parts of a computation, even when the computation is split over multiple machines that don't share a common file system. Different users can

also use the virtual file system to collaborate, sharing data files and even accessing the same running computations.

K. Interoperability

The standard Legion host object creates objects using the process creation interface of the underlying operating system. However, some systems require using a queue management system to take full advantage of local resources. For example, some parallel computers contain a small number of "interactive" nodes, which can be accessed through normal means, and a large number of "compute" nodes, which can only be reached by submitting jobs to a local queue management system.
To make use of hosts that are managed by local queuing systems, Legion provides a modified host object implementation called the BatchQueueHost. BatchQueueHost objects submit jobs to the local queuing system, instead of using the standard process creation interface of the underlying operating system. A BatchQueueHost can be used with a variety of queue systems. **LoadLeveler, Codine, PBS, and NQS** are the currently supported queue types.

Major requirements

1. Availability and quality of:

a. binary code	***Yes***
b. source code	***Yes***
c. documentation	***Yes***
d. roadmap	***No info***
e. Training	***No***
f. customer support	***No***

Score: *3*

2. Operating systems/platforms

a. submission	***UNIX/NT***
b. execution	***UNIX/NT***
c. control	***UNIX/NT***

Supported architectures:

solaris (Sun workstations running Solaris 5.x)
sgi (SGI workstations running IRIX 6.4)
linux (x86 running Red Hat 5.x Linux)
x86_freebsd (x86 running FreeBSD 3.0)
alpha_linux (DEC Alphas running Red Hat 5.x Linux)
alpha_DEC (DEC Alphas running OSF1 v4)
rs6000 (IBM RS/6000s running AIX 4.2.1)
hppa_hpux (HPUX 11)
winnt_x86 (Windows NT 4.0)
t90 (Cray T90s running Unicos 10.x virtual hosts only)
t3e (Cray T3E running Unicos/mk 2.x virtual hosts only)

Score: *3.5 (NT is a beta release)*

3. Batch job support

Score: *4*

4. Interactive job support

TTY objects.

Score: *4*

5. **Parallel job support**

Legion offers the following parallel support:

Support of parallel libraries

The vast majority of parallel applications today are written in MPI and PVM. Legion supports both libraries, via emulation libraries that use the underlying Legion run-time library. Existing applications only need to be recompiled and relinked in order to run on Legion. MPI and PVM users can thus reap the benefits of Legion with existing applications. In the future, libraries such as Scalapak will also be supported.

Parallel language support

Legion supports MPL (Mentat Programming Language) and BFS (Basic Fortran Support). MPL is a parallel C++ language in which the user specifies those classes that are computationally complex enough to warrant parallel execution. Class instances are then used like C++ class instances: the compiler and run-time system take over and construct parallel computation graphs of the program and then execute the methods in parallel on different processors. Legion is written in MPL. BFS is a set of pseudo-comments for Fortran and a preprocessor that gives the Fortran programmer access to Legion objects. It also allows parallel execution via remote asynchronous procedure calls and the construction of program graphs. HPF may also be supported in the future.

Wrap parallel components

Object wrapping is a time-honored tradition in the object-oriented world. We have extended the notion of encapsulating existing legacy codes into objects by encapsulating parallel components into objects. To other Legion objects the encapsulated object appears sequential but it executes faster. PVM, HPF, and shared memory threaded applications can thus be encapsulated into a Legion object.

Export the run-time library

We do not expect to provide the full range of languages and tools that users require: instead of developing everything here at the University of Virginia, we anticipate Legion becoming an open, community, artifact, to which other tools and languages are ported. To support these third party developments, the complete run-time library is available. User libraries can directly manipulate the run-time library.

The library is completely re-configurable. It supports basic communication, encryption/decryption, authentication, and exception detection and propagation, as well as parallel program graphs. Program graphs represent functions and are first class and recursive. Graph nodes are member function invocations on Legion objects or sub-graphs. Arcs model data dependencies. Graphs may be annotated

with arbitrary information, such as resource requirements, architecture affinities, etc. Schedulers, fault-tolerance protocols, and other user-defined services may use the annotations.

Score: *2 (No information about the library itself)*

6. Resource requests by users

Just host and vault objects can be requested. No info about host attributes.

Score: *0*

7. Limits on resources by administrators

a. general ***No info.***

b. specific users or groups ***No info.***

Score: *0*

8. File stage-in and stage-out

Legion virtual file system (uses vault objects)

Score: *4*

9. Flexible scheduling

a. several scheduling policies

Just one default scheduler.

b. policies changing in time

No.

c. scheduling multiple resources simultaneously

No info.

d. highly configurable scheduling

Ability to develop per-application schedulers.
Ability to submit jobs to other schedulers.

Score: *1*

10. Job priorities

No info.

Score: *0*

11. Timesharing

a. processes

Yes

b. jobs
Yes

Score: *4*

12. Impact on owners of computational nodes
Minimal impact on computational nodes.

Score: *4*

13. Checkpointing
- **a. user-level** — ***Just for MPI programs.***
- **b. run-time library level** — ***No***
- **c. OS (kernel) level** — ***No***

Score: *0.5*

14. Suspending/resuming/killing jobs
- **a. user** — ***Yes***
- **b. administrator** — ***Probably by local admin***
- **c. Automatic** — ***No info***

Score: *2*

15. Process migration
- **a. user** — ***No***
- **b. administrator** — ***No***
- **c. automatic** — ***No***

Score: *0*

16. Static load balancing
No info.

Score: *0*

17. Dynamic load balancing
No info.

Score: *0*

18. Fault tolerance
a. **computational nodes**
MPI checkpointing
b. **scheduler**
No central scheduler

Score: *0*

19. Job monitoring
No info.

Score: *0*

20. User interface
CLI.

Score: *4*

21. Published APIs
MPL

Score: *3*

22. Dynamic system reconfiguration
Yes.

Score: *4*

23. Security

a. authentication	***Kerberos***
b. authorization	***ACL on every object***
c. encryption	***Yes***

Score: *4*

24. Accounting
No info.

Score: *0*

25. Scalability

Score: *4*

TOTAL SCORE = ***51***

Major advantages:
- completly distributed
- scalable
- easy to augment
- can submit jobs to other schedulers

Major disadvantages:

- doesn't have a proper scheduler
- weak resource specification

Resource:

http://www.cs.virginia.edu/~legion/

Legion 1.7 Basic User Manual
Legion 1.7 Developer Manual
Legion 1.7 Reference Manual
Legion 1.7 System Administrator Manual

A.10 NetSolve

Background information

A. Authors
Jack Dongarra, Innovative Computing Laboratory, University of Tennessee

B. Support
Innovative Computing Laboratory, University of Tennessee

C. Distribution
Public domain

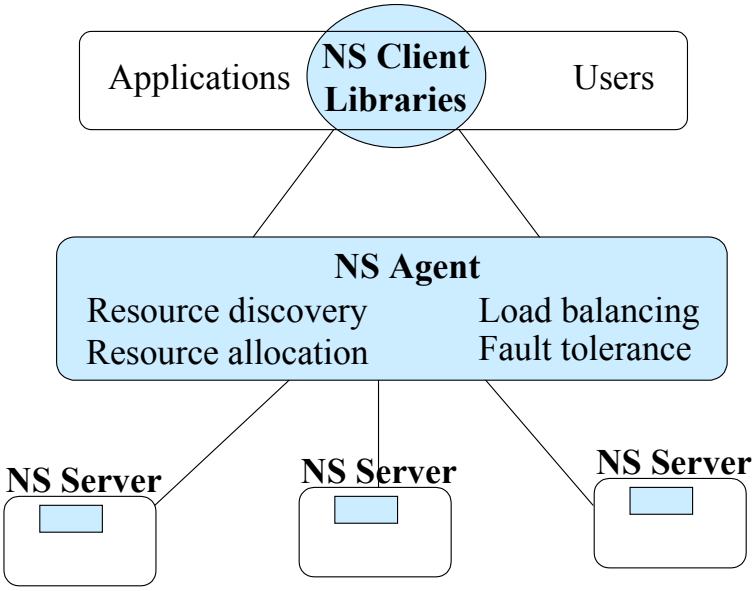
D. License

System Description

E. Type of the system
distributed Job Management System without a central job scheduler

F. Major modules
NS Client - User Module
NS Agent - Resource Manager
NS Server - Execution host deamon, part of Resource Manager

G. Functional flow



NetSolve client library is linked in with the user's application. The application then makes calls to NetSolve's application processing interface (API) for specific services. Through the API, NetSolve client-users gain access to aggregate resources without the users needing to know anything about computer networking or distributed computing.
NetSolve agent maintains a database of NetSolve servers along with their capabilities (hardware performance and allocated software) and dynamic usage statistics. It uses this

information to allocate server resources for client requests. The agent, in its resource allocation mechanism, attempts to find the server that will service the report the quickest, balance the load among its servers, and keep track of failed servers. Requests are directed away from failed servers.
NetSolve server is a daemon process that awaits client requests. The server can run on single workstations, symmetric multi-processors, or machines with massively parallel processors.

The functional flow is as follows:

1. Client contacts the agent for a list of capable servers.
2. Client contacts server and sends input parameters
3. Server runs appropriate service
4. Server returns output parameters or error status to client.

H. Distribution of control
Distributed control.

I. Distribution of information services
Distributed information services

J. Communication

a. Protocols
XDR (External Data Representation Standard)

H. Interoperability

a. running as a part of another system
b. JMS using another system in place of its module
c. JMS running concurrently with another system

Major requirements

1. Availability and quality of

a. binary code	Yes
b. source code	Yes
c. documentation	Good
d. roadmap	Yes Java GUI, Microsft Excel interface, enhanced load balancing
e. training	No
f. customer support	No

Score: 3

2. Operating systems/platforms

a. Submission
b. Execution
c. Control

Unix-like, Windows

Score: 4

3. Batch job support No

Score: 0

4. Interactive job support Yes

Score: 4

5. Parallel job support

Score: 0

6. Resource requests by users

Score: 0

7. Limits on resources by administrators
a. general
b. specific users or groups

Score: 0

8. File stage-in and stage-out

Score: 0

9. Flexible scheduling
a. several scheduling policies
b. policies changing in time
c. scheduling multiple resources simultaneously
d. highly configurable scheduling

Score: 0

10. Job priorities
a. user assigned
b. system assigned

Score: 0

11. Timesharing
a. processes
b. jobs

Score: 2

12. Impact on owners of computational nodes

Medium

Score: 2

13. Checkpointing
- **a. User-level**
- **b. run-time library level**
- **c. OS (kernel) level**

Score: 0

14. Suspending/resuming/killing jobs
- **a. user**
- **b. administrator**
- **c. automatic**

Score: 4

15. Process migration
- **a. user**
- **b. administrator**
- **c. automatic**

Score: 1

16. Static load balancing

Score: 4

17. Dynamic load balancing

Score: 0

18. Faul tolerance
- **a. computational nodes**
- **b. scheduler**

Score: 4

19. Job monitoring
- **a. real-time monitoring tools (CLI and GUI)**
- **b. history logs**

Score: 0

20. User interface
- **a. CLI**
- **b. GUI**

Score: 4

21. Published APIs

Score: 4

22. Dynamic system reconfiguration

Score: 4

23. Security
- **a. authentication**
- **b. authorization**
- **c. encryption**

Score: 0

24. Accounting

Score: 0

25. Scalability

Score: 4

TOTAL SCORE = 44

Major Disadvantages

No support for batch jobs
No queuing system, no central scheduler
No support for parallel jobs
Limited set of tasks executed by the computational nodes.

Major Advantages

User friendly - Job management transparent to users
Interfaces with Matlab and Java.

References

NetSolve Homepage:
http://www.cs.utk.edu/netsolve/

H. Casanova and J. Dongarra, "Providing Access to High Performance Computing Technologies," Springer-Verlag's Lecture Notes in Computer Science #1184, pp. 123-134.

H. Casanova and J. Dongarra, "NetSolve: A Network Server for Solving Computational Science Problems," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, No. 3, pp. 212-223, Fall 1997.

D. C. Arnold, W. Lee, J. Dongarra, and M. Wheeler, "Providing Infrastructure and Interface to High Performance Applications in a Distributed Setting," *High Performance Computing 2000*.

H. Casanova, J. Plank, M. Beck, and J. Dongarra, "Deploying Fault-Tolerance and Task Migration with NetSolve," to appear in the *International Journal on Future Generation Computer Systems*.

A.11. The AppLES Parameter sweep template

Background information

A. Authors
Henri Casanova, Francine Berman, Graziano Obertelli: Computer Science and Engineering Department University of California, San Diego
Richard Wolski: Computer Science Departament University of Tennessee, Knoxwille

B. Support
University of California, San Diego, NSF, NASA, DoD, DARPA

C. Distribution (commercial/public domain)
Free

D. License
Unknown

System description

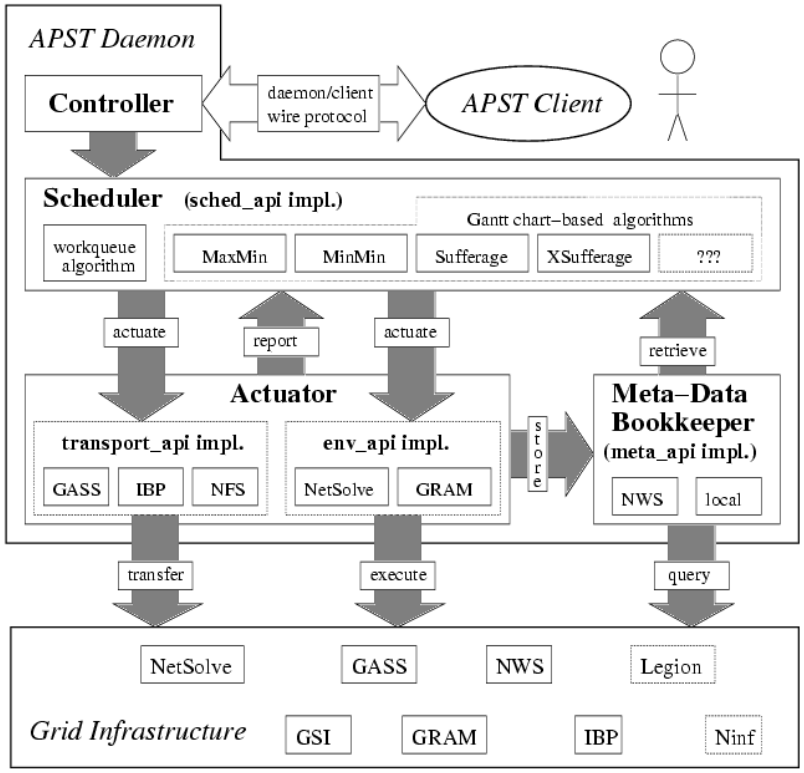
E. Type of the system
User-level middleware, user-level scheduler for parameter-sweep applications over large parameter spaces (*application-level data dispatcher)*

F. Major modules

- ***Controller*** *(user server)*
- ***Scheduler***
- ***Actuator*** *(job dispatcher)*
- ***Meta-data bookkeeper*** *(resource monitor)*

No queuing system

G. Functional flow



- ***Controller*** *(user server)*
  Relays information between the client and the daemon and notifies the scheduler of new tasks to perform or task cancellations. Uses the scheduler's API and communicates with the client using a simple wire protocol.
- ***Scheduler***
  The central component of the APST daemon. Its API is used for notification of events concerning the application's structure (new tasks, task cancellations), the status of computational resources, (new disk, new host, host/disk/network failures) and the status of running tasks (task completitions or failures). The behavior of the scheduler is entirely defined by the implementation of this API.
- ***Actuator*** *(job dispatcher)*
  Implements all interaction with the grid infrastructure software for accessing storage, network and computational resources. It also interacts with the grid security infrastructure on behalf of the user when needed.
- ***Meta-data bookkeeper*** *(resource monitor)*
  In charge of keeping track of static and dynamic meta-data concerning both the resources and the application. It is also responsible for performing or obtaining forecasts for various types of metadata.
- ***Grid infrastructure***
  Acts as itself

H. Distribution of control

Centrally located user level scheduler interacting with various JMSes and information services

I. Distribution of information services

Information about available resources is collected from distributed information systems and stored in *Bokkeeper* module

J. Communication

a. Protocols

TCP/IP, higher-level protocols depend on other systems

b. Shared file systems

None

K. Interoperability

a. JMS running as a part of another systems

None

b. JMS using another system in place of its module

NetSolve, Globus, NWS, IBP, Legion, Ninf, Condor

c. JMS running concurrently with another system

None

Major requirements

1. Availability

a. Binary code	Yes
b. Source code	Yes
c. Documentation	Good (user documentation, publications)
d. Roadmap	Allow APST daemon and APST client to reside on different file systems, develop better long-range forecasting techniques, integration of further parts of Globus toolkit, development of implementations of APST Actuator over other Grid softwares like Ninf and Legion.
e. Training	No
f. Customer support	No
Score:	*3*

2. Operating systems/platforms

a. Submission	*NIX
b. Execution	depends on underlying JMS
c. Control	*NIX

The software has been developed on Linux and ported to most UNIX platforms

Score: *1*

3. Batch job support

No (no general queuing system), but may depend on user interface

Score: *2*

4. Interactive job support

Not clearly mentioned, but may support (depends on UI)

Score: *2*

5. Parallel job support

Yes

Score: *2*

6. **Resource requests by users**

Score: *2*

7. Limits on resources by administrators

a. General No

b. Specific users or groups No

Score: *0*

8. File stage-in and stage-out

Done by the user

Score: *0*

9. Flexible scheduling

a. Several scheduling policies Yes

b. Policies changing in time Yes

c. Scheduling multiple resources simultaneously Yes

d. Highly configurable scheduling No

Score: *2*

10. Job priorities

Score: *0*

11. **Timesharing**

a. Processes No

c. Jobs No

Score: *0*

12. Impact on owners of computational nodes

a. Configurability

Depends on underlying JMS

b. Influence

Depends on underlying JMS

Score: *0*

13. Check pointing

a. User-level No

b. Run-time library level No

c. OS (kernel) level No

Score: *0*

14. Suspending/resuming/killing jobs

a. User	No
b. Administrator	No
c. Automatic	No
Score:	*0*

15. Process migration

Probably possible rescheduling

a. User	No
b. Administrator	No
c. Automatic	Yes
Score:	*1*

16. Static load balancing

Score: *4*

17. Dynamic load balancing

Probably no

Score: *0*

18. Fault tolerance

a. Computational nodes	Depends on underlying JMS
b. Scheduler	No
Score:	*0*

19. Job monitoring

Unknown

Score: *0*

20. User interface

Files with tasks descriptions

Score: *1*

21. Published APIs

Score: *4*

22. Dynamic system reconfiguration

Score: *4*

23. **Security**

a. Authentication	No
b. Authorization	No
c. Encryption	No
Score:	*0*

24. Accounting
Done by JMS
Score: *0*

25. Scalability
Depends on underlying JMS
Score: *2*

MAXIMUM TOTAL SCORE = **30**

Advantages

- Cooperation with other systems
- User-defined scheduling policies

Disadvantages

- Not a JMS
- Submissions only by a single user
- No Windows NT support
- No security mechanisms
- No accounting capabilities
- Poor user interface

References

Homepage: `http://apples.ucsd.edu/`

Henri Casanova, Graziano Obertelli, Francine Berman and Rich Wolski: "*The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid*", Proceedings of the Super Computing Conference (SC'2000), `http://gcl.ucsd.edu/apst/publications/apst_sc00.ps`

A.12 Network Weather Service (NWS)

Basic features

A. Authors
Rich Wolski, Jim Hayes (University of California, San Diego), Martin Swany (University of Tennessee, Knoxwille)

B. Support
National Partnership for Advanced Computational Infrastructure

C. Distribution (commercial/public domain)
Public Domain

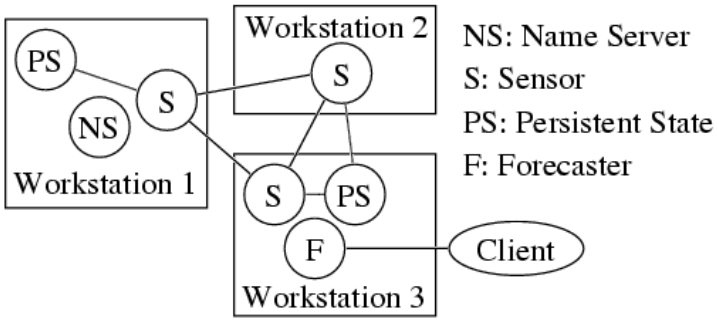
D. License

System description

E. Type of the system
Distributed resource monitor
The Network Weather Service is a distributed system that periodically monitors and dynamically forecasts the performance various network and computational resources can deliver over a given time interval. The service operates a distributed set of performance sensors (network monitors, CPU monitors, etc.) from which it gathers readings of the instantaneous conditions. It then uses numerical models to generate forecasts of what the conditions will be for a given time frame.

F. Major modules
- Persistent State process
- Name Server process
- Sensor process (*resource monitor)*
- Forecaster process

G. Functional flow



- **Persistent State** process: stores and retrieves measurements from persistent storage.
- **Name Server** process: implements a directory capability used to bind process and data names with low-level contact information (e.g. TCP/IP port number, address pairs).
- **Sensor** process: gathers performance measurements from a specified resource.

- **Forecaster** process: produces a predicted value of deliverable performance during a specified time frame for a specified resource.

H. Distribution of control
Clients contact centrally-located forecaster process

I. Distribution of information services
Information is stored by *Persistent state* processes on each node

J. Communication architecture

a. Protocols
TCP/IP and LDAP

b. Shared file systems
None

K. Interoperability

a. JMS running as a part of another
Globus, Condor, AppLES

b. JMS using another system in place of its module
No

c. JMS running concurrently with another system
Any JMS

Major requirements

1. **Availability and quality of**
   - **a. Binary code** Available
   - **b. Source code** Available
   - **c. Documentation** Average (research papers)
   - **d. Roadmap** Support for 64-bit architectures,
     Ability to make forecasts from measurements that are not stored in an NWS memory,
     Automatic configuration of clique topology, improved persistence of NWS hosts, especially the name server,
     Basing the NWS name server on LDAP,
     Separating the clique protocol from the NWS so that it can be used by other client codes,
     Improved CPU sensor
   - **e. Training** Not available
   - **f. Customer support** Not available

   Score: *3*

2. **Operating systems/platforms**
   - **a. Submission** *NIX
   - **b. Execution** *NIX
   - **c. Control** *NIX

   Score: *3*

3. **Batch job support**
   Not a JMS
   Score: *0*

4. **Interactive job support**
   Not a JMS
   Score: *0*

5. **Parallel job support**
   Not a JMS
   Score: *0*

6. **Resource requests by users**
   Does not have resource manager
   Score: *0*

7. **Limits on resources by administrators**
   - **a. General** None
   - **b. Specific users or groups** None

   Score: *0*

8. File stage-in and stage-out

Not supported

Score: *0*

9. Flexible scheduling

Not a JMS

a. Several scheduling policies Not supported

b. Policies changing in time Not supported

c. Scheduling multiple resources simultaneously Not supported

e. Highly configurable scheduling Not supported

Score: *0*

10. Job priorities

Not supported (not a JMS)

Score: *0*

11. Timesharing

Not supported (not a JMS)

a. Processes None

b. Jobs None

Score: *0*

12. Impact on owners of computational nodes

a. Configurability: No

b. Influence Minimal

Score: *4*

13. Check pointing

Not supported (not a JMS)

a. User-level None

b. Run-time library level None

c. OS (kernel) level None

Score: *0*

14. Suspending/resuming/killing jobs

Not supported (not a JMS)

a. User No

b. Administrator No

c. Automatic No

Score: *0*

15. Process migration

Not supported (not a JMS)

a. User No

b. Administrator No

c. Automatic No
Score: *0*

16. Static load balancing
Not supported (not a JMS)
Score: *0*

17. Dynamic load balancing
Not supported (not a JMS)
Score: *0*

18. Fault tolerance
a. Sensors Yes
b. Forecaster No
c. Name server No
Score: *1*

19. Job monitoring
Not supported (not a JMS)
Score: *0*

20. User interface
CGI Web interface
Score: *2*

21. Published APIs
Available C API
Score: *4*

22. Dynamic system reconfiguration
Score: *4*

23. Security
a. Authentication Not applicable
b. Authorization Not applicable
c. Encryption Not available
Score: *0*

24. Accounting
Not applicable
Score: *0*

25. Scalability
Score: *4*

MAXIMUM TOTAL SCORE = **25**

Advantages	Disadvantages
• Excellent resource usage forecasting • May be used as a part of a JMS	• Not a JMS • No Windows NT support

References:

Homepage: `http://nws.npaci.edu/NWS/`

Rich Wolski, Neil T. Spring, and Jim Hayes, The Network Weather Service: "*A Distributed Resource Performance Forecasting Service for Metacomputing*", Journal of Future Generation Computing Systems, 1999;
`http://www.cs.ucsd.edu/users/rich/papers/nws-arch.ps.gz\`

A.13 Compaq DCE

Background information

A. Authors

Open Software Foundation (OSF)
Other authors and implementations of DCE:
Compaq (Digital) for Windows NT, True64 Unix, Open VMS,
Gradient/Entegrity Company - NetCrusader, PC-DCE and SysV-DCE
IBM DCE
Transarc Company - DFS and Encina
Hewlett-Packard's

B. Support

Open Group

C. Distribution (commercial/public domain)

Commercial/Free

D. License

Open commercial

System description

E. Type of the system

Standard environment for support of distributed applications DCE provides a complete Distributed Computing Environment infrastructure. It provides security services to protect and control access to data, name services that make it easy to find distributed resources, and a highly scalable model for organizing widely scattered users, services, and data.

DCE is called "middleware" or "enabling technology." It is not intended to exist alone, but instead should be bundled into a vendor's operating system offering, or integrated in the third-party applications. DCE's security and distributed filesystem, for example, can completely replace their current, non-network, analogs. DCE is not an application in itself, but is used to build custom applications or to support purchased applications.

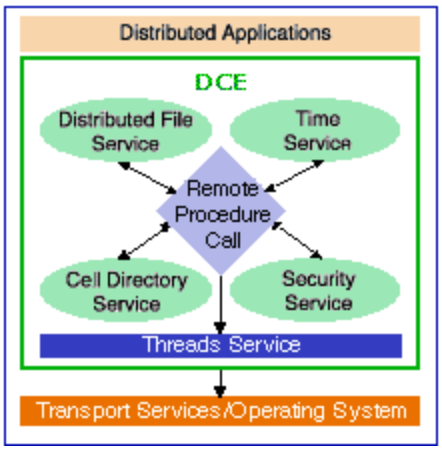
F. Major modules

DCE is constructed on a layered architecture, from the most basic providers of services (e.g., operating systems) up to higher-end clients of services (e.g., applications). Security and management are essential to all layers in the model. Currently, DCE consists of seven tools and services that are divided into fundamental distributed services and data-sharing services.

The fundamental distributed services include threads, RPCs, directory service, time service, and security service. Data-Sharing Services build on top of the fundamental services and include DFS (Distributed File System) and diskless support. The OSF

has reserved space for possible future services, such as spooling, transaction services, and distributed object-oriented environments.

G. Functional flow



DCE provides a communications environment that supports information flow from wherever it's stored to wherever it's needed, without exposing the network's complexity to the end-user, system administrator or application developer. The DCE architecture masks the physical complexity of the networked environment by providing a set of services that can be used—separately or in combination—to form a comprehensive distributed computing environment.
The DCE architecture (see Figure 1) is a layered model that integrates a set of eight fundamental technologies bottom up from the most basic supplier of services (e.g., the operating system) to the highest level consumers of services (e.g., applications). Security and management services are essential to all layers of the environment. To applications, DCE appears as a single logical system, which can be organized into two broad categories of services: the DCE Secure Core and DCE Data Sharing Services.

H. Distribution of control

N/A

I. Distribution of information services

Depends on implementation. All services can be implemented as distributed

J. Communication

a. Protocols

TCP, UDP, DCE RPC, LDAP, MIT Kerberos 5

b. Shared file systems

DFS

K. Interoperability

a. JMS running as a part of another systems

Can provide authentication and authorization services to JMSes, directory services, time services and distributed file system.

b. JMS using another system in place of its module

No

c. JMS running concurrently with another system

Yes

Major requirements

1. Availability

a. Binary code	Yes
b. Source code	Yes
c. Documentation	Good (is to be purchased)
d. Roadmap	Unknown
e. Training	Yes
f. Customer support	Yes
Score:	*4*

2. Operating systems/platforms

a. Submission	Solaris, Windows NT, True64UNIX
b. Execution	N/A
c. Control	Solaris, Windows NT, True64UNIX
Score:	*3*

3. Batch job support

Not a JMS

Score: *0*

4. Interactive job support

Not a JMS

Score: *0*

5. Parallel job support

Not a JMS

Score: *0*

6. **Resource requests by users**

Not a JMS

Score: *0*

7. Limits on resources by administrators

Not a JMS

a. General	No
b. Specific users or groups	No
Score:	*0*

8. File stage-in and stage-out

Not a JMS

Score: *0*

9. Flexible scheduling

Not a JMS

a. Several scheduling policies	No
b. Policies changing in time	No
c. Scheduling multiple resources simultaneously	No
d. Highly configurable scheduling	No
Score:	*0*

10. Job priorities

Not a JMS

Score: *0*

11. Timesharing

Not a JMS

a. Processes	No
b. Jobs	No
Score:	*0*

12. Impact on owners of computational nodes

Depends on implementation

a. Configurability	Unknown
b. Influence	Unknown
Score:	*0*

13. Check pointing

Not a JMS

a. User-level	N/A
b. Run-time library level	N/A
c. OS (kernel) level	N/A
Score:	*0*

14. Suspending/resuming/killing jobs

Not a JMS

a. User	N/A
b. Administrator	N/A
c. Automatic	N/A
Score:	*0*

15. Process migration

Not a JMS

a. User	N/A
b. Administrator	N/A
c. Automatic	N/A

Score: *0*

16. Static load balancing

Not a JMS

Score: *4*

17. Dynamic load balancing

Not a JMS

Score: *0*

18. Fault tolerance

a. Computational nodes Yes

b. Scheduler N/A

Score: *4*

19. Job monitoring

Not a JMS, but provides tools that can be used to monitor the jobs and events.

Score: *2*

20. User interface

Depends on implementation

Score: *2*

21. Published APIs

Score: *4*

22. Dynamic system reconfiguration

Score: *4*

23. Security

a. Authentication Public key certificates

b. Authorization Access Control Lists (ACLs)

c. Encryption Yes

Score: *4*

24. Accounting

Unknown

Score: *0*

25. Scalability

Good scalability.

Score: *4*

MAXIMUM TOTAL SCORE = **35**

Advantages	Disadvantages
• Strong authentication and authorization techniques • Distributed file system • Distributed directory services • Distributed time service	• Not a JMS

References

DCE Product Overview
http://www.transarc.ibm.com/Product/DCE/DCEOverview/dceoverview.html

Michael D. Millikin *"DCE: Building the Distributed Future"*, BYTE June 1994,
http://www.byte.com/art/9406/sec8/art1.htm

DCE Frequently Asked Questions, http://www.opengroup.org/dce/info/faq-mauney.html

Compaq DCE :
http://www.tru64unix.compaq.com/dce/index.html